NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

SOLVING THE MAXIMUM CLIQUE PROBLEM ON A CLASS OF NETWORK GRAPHS, WITH APPLICATION TO SOCIAL NETWORKS

by

Spyridon Pollatos

June 2008

Thesis Advisors:	W. Matthew Carlyle
	Ralucca Gera
Second Reader:	Pantelimon Stanica

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Solving the Maximum Clique Problem on a Class of Network Graphs, with Application to Social Networks			**5. FUNDING NUMBERS**
6. AUTHOR(S) Spyridon Pollatos			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Social network analysis frequently uses the idea of a clique in a network to identify key subgroups of highly-connected members of the network. We formulate the maximum clique problem on undirected graphs and develop two algorithms to solve it: a pruning algorithm and an enumeration algorithm. The pruning algorithm successively improves an upper bound on the clique number of a graph, and the enumeration algorithm successively finds larger and larger cliques in the graph. Both terminate with a maximum clique in the graph, and, when run together, provide an interval of uncertainty on the size of a maximum clique in a graph that converges to zero. We apply our algorithms to real examples in the modeling of terrorist social networks, and determine that our algorithms are efficient and practical for problems of moderate size.

14. SUBJECT TERMS Maximum Clique, Clique Number, Graph, Backtracking algorithm, Social Network, Terrorist Network.			**15. NUMBER OF PAGES** 93
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SOLVING THE MAXIMUM CLIQUE PROBLEM ON A CLASS OF NETWORK GRAPHS, WITH APPLICATION TO SOCIAL NETWORKS

Spyridon Pollatos
Lieutenant Commander, Hellenic Navy
B.S., Hellenic Naval Academy, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH
and
MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2008

Author:	Spyridon Pollatos
Approved by:	W. Matthew Carlyle Thesis Co-Advisor
	Ralucca Gera Thesis Co-Advisor
	Pantelimon Stanica Second Reader
	James N. Eagle Chairman, Department of Operations Research
	Clyde Scandrett Chairman, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Social network analysis frequently uses the idea of a clique in a network to identify key subgroups of highly-connected members of the network. We formulate the maximum clique problem on undirected graphs and develop two algorithms to solve it: a pruning algorithm and an enumeration algorithm. The pruning algorithm successively improves an upper bound on the clique number of a graph, and the enumeration algorithm successively finds larger and larger cliques in the graph. Both terminate with a maximum clique in the graph, and, when run together, provide an interval of uncertainty on the size of a maximum clique in a graph that converges to zero. We apply our algorithms to real examples in the modeling of terrorist social networks, and determine that our algorithms are efficient and practical for problems of moderate size.

THIS PAGE INTENTIONALLY LEFT BLANK

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

DIMACS	Discrete Mathematics and theoretical Computer Science
GWOT	Global War On Terrorism
LIFO	Last In First Out
MCP	Maximum Clique Problem
MISP	Maximum Independent Set Problem
NP	Non-deterministic Polynomial time
OR	Operations Research
P	Polynomial time
SNA	Social Networks Analysis

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

An undirected, simple graph, $G=(V,E)$, is defined by a set of *vertices, V*, and a set of *edges*, *E*, consisting of unordered pairs of distinct vertices that represent symmetric, pairwise relationships, or *adjacencies*, between those vertices. A *clique* in a graph *G* is a subset of vertices, $C \subseteq V$, such that every pair of vertices in *C* corresponds to an edge in *E*. Namely, every pair of vertices in *C* must be adjacent in *G*, and so *C* induces a *complete*, or *maximally connected*, subgraph of *G*. Finding cliques of maximum cardinality in a graph is a long-standing problem in graph theory, and is referred to as the *maximum clique problem*, or MCP.

The MCP has important applications in many different domains. Some examples of military applications are in Cryptography and Cryptanalysis, Telecommunications, particularly in Wireless Networks and in Radio Frequency Assignment, and Social Networks Analysis. Since the clique is considered as the foundational idea for studying cohesive subgroups in social networks, this thesis considers the maximum clique problem with an emphasis on its application to military problems, especially in Social Networks Analysis.

The primary question that this thesis addresses is the following:

Can we identify and implement an exact algorithm to solve the Maximum Clique Problem (MCP) on undirected graphs, in a reasonable time frame?

To achieve the above objective we selected two algorithms from the current literature and modified them to improve their performance, developing the *pruning* and *enumeration* algorithms. We then implemented both of these in a modern, powerful and widely used programming language, Java. Our testing involved applying both algorithms to real-world situations which could be modeled by undirected graphs: terrorist social networks.

We verified that both algorithms solve the MCP on undirected graphs, and are quick on relatively small graphs. Furthermore, the pruning algorithm immediately

establishes an upper bound on the clique number of a graph and then successively improves this bound; thus the pruning algorithm can be terminated early with a valid upper bound on the clique number. Similarly, the enumeration algorithm quickly finds small cliques, and successively discovers larger and larger cliques in the graph as it progresses, each of which provides a lower bound on the clique number of the graph. It can be terminated early with a valid lower bound on the clique number.

If both are run simultaneously each provides a bound the other cannot, and an interval of uncertainty can be established that will eventually be reduced to zero, at which point a maximum clique will have been found.

Moreover analyzing a social network using the Social Networks Analysis methods and measures maximum cliques and vertices degree may provide enough information about the social structure of the network under investigation. More particularly, the maximum clique(s) and the clique(s) as well, may give us the most cohesive subgroups while the vertices with the largest vertex-degrees may show the most central actors in a social network. However, since cliques have been criticized for their restrictive nature we may consider and study one of the branches of the relaxation clique problem and analyze a social network under this concept. This may be the subject for an extension of this current thesis and an area for further investigation. It is worthwhile to point out that each algorithm can be modified in fairly straightforward ways to allow various relaxations of the definition of a clique.

ACKNOWLEDGMENTS

I would like to express my deepest gratefulness to my thesis advisors Dr. Ralucca Gera and Dr. W. Matthew Carlyle as well as to my second reader Dr. Pantelimon Stanica for their guidance and contribution to my thesis work with their encouragement, expert knowledge, and invaluable advices. I also want to acknowledge and heartily thank my wife Elisavet and my two daughters Stavroula and Anastasia for their love, patience, understanding and support to my efforts.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

It has been said that figures rule the world. Maybe. But I am sure that figures show us whether it is being ruled well or badly.

Goethe

A. CLIQUES IN SOCIAL NETWORK ANALYSIS

An undirected, simple graph, $G=(V,E)$, is defined by a set of *vertices*, V, and a set of *edges*, E, consisting of unordered pairs of distinct vertices that represent symmetric, pairwise relationships, or *adjacencies,* between those vertices. A *clique* in a graph G is a subset of vertices, $C \subseteq V$, such that every pair of vertices in C corresponds to an edge in E. Namely, every pair of vertices in C must be adjacent in G, and so C induces a *complete*, or *maximally connected*, subgraph of G. Finding cliques of maximum cardinality in a graph is a long-standing problem in graph theory, and is referred to as the *maximum clique problem*, or MCP.

The MCP has important applications in many different domains. Some examples of military applications are in Cryptography and Cryptanalysis, Telecommunications, particularly in Wireless Networks and in Radio Frequency Assignment, and Social Networks Analysis. In fact, according to Wasserman and Faust [10], “The clique is the foundational idea for studying cohesive subgroups in social networks….” This thesis considers the maximum clique problem with an emphasis on its application to military problems, especially in Social Networks Analysis.

B. THESIS OBJECTIVE AND RESEARCH QUESTION

The primary question that this thesis addresses is the following:

Can we identify and implement an exact algorithm to solve the Maximum Clique Problem (MCP) on undirected graphs, in a reasonable time frame?

To achieve the above objective we first review the current literature on MCP, and then select two existing algorithms related to this problem. We implement efficient versions of both and apply them to real-world situations. We focus on social networks, where we find a wide variety of important applications. We will also consider interesting variations to the initial problem, such as various relaxations of the clique requirement that overcome objections to the MCP's restrictive nature and modeling disadvantages. Finally, we analyze the behavior of our two algorithms and draw conclusions on their performance, utility and effectiveness. Our main intention is to improve their performance and obtain a new and more efficient algorithms that can be applied to military applications of the maximum clique problem (and its relaxations) for problems of reasonable size.

C. LITERATURE REVIEW AND ITS CONNECTION TO CURRENT WORK

The decision version of the MCP, in which the goal is to decide whether a given graph contains a clique of a given cardinality, is one of the first problems that have been proved to be NP-complete (see [1] for history and discussion), and therefore the MCP is an NP-Hard problem.

Tarjan [7] was one of the first who addressed the MCP and presented one of the first algorithms that gave a reasonable and effective solution to the problem at hand. Tarjan provided an algorithm with a running time bounded by $O\left(n2^n\right)$, where n is the order of a given graph G. The basic algorithm examines every subset of the vertex set $V\left(G\right)$ of a given a graph G with n vertices. The algorithm determines all subsets that are cliques, and chooses the largest clique found as the maximum clique. Since the number of subsets of any set with n elements is 2^n, and it takes $O\left(n\right)$ time to check if a subset forms a clique, it follows that the time upper bound for this simple algorithm to solve the MCP is $O\left(n2^n\right)$.

Tarjan later succeeded in improving this basic algorithm and discovered other, improved algorithms. The first one had a worst-case time bound of $k\left(1.286\right)^n$ for some constant k. Hence, within a fixed amount of time this improved algorithm could analyze a larger graph than the basic algorithm. A few years later, Tarjan and Trojanowski [13] presented a recursive algorithm which determines a maximum independent set of n-vertex graph in $O\left(2^{n/3}\right)$ time (see also [12]).

Since the early 1970s, many papers have been published with algorithms for the MCP [2]. According to L. Babel in [3], earlier work diverged into two directions. The first concerned algorithms solving the problem for arbitrary graphs in exponential time, the other restricted to special classes of graphs where polynomial methods could be found.

Between September 1992 and September 1993, the Second DIMACS Implementation Challenge took place. The purpose of the challenge was to encourage high-quality research on empirical issues in combinatorial optimization. The problem of finding cliques in graphs was one of the three problem classes that was discussed. According to the analytical results of this challenge, which were presented in [4], it seems unlikely to have a fast, i.e., polynomial time, algorithm to solve this kind of problem exactly. Even finding an approximate solution quickly is improbable due to the fact that it is an NP-complete problem. Moreover, all the papers presented during this challenge on finding cliques in a graph were a mixture of exact and heuristic methods, that is a mixture of exact and approximate methods [4]. We present a brief description of these two algorithm categories.

The exact algorithms are those that have been proven mathematically to provide an optimal solution. Branch-and-bound, for example, is a finite computing time method that has been widely implemented in the efforts of solving the MCP or a part of it [2, 4]. Besides the quality of the solution that this method guarantees, it also provides the solution in an acceptable finite computing time.

On the other hand, heuristics algorithms are those that cannot guarantee any solution quality. Greedy algorithms, Neighborhood search, and Tabu search are typical examples of heuristic methods which also have been considered in approaching the MCP [4, 9].

Early algorithms for solving the MCP as well as recent approaches included the branch-and-bound method. In [12], Wood referred to many fundamental approaches to the maximum clique problem which include branch-and-bound algorithms. First of all, Wood considered, in this paper, the significance of determining an upper and a lower bound for problems that are NP-complete like the MCP. He also presented a branch-and-bound algorithm for finding a maximum clique in a graph, distinguishing two algorithms which were the most efficient ones known for the maximum clique problem until 1997. One of these algorithms was developed by Babel and presented in [3] while the other belongs to Balas and Xue and was presented in [14]. These algorithms calculated lower and upper bounds, which seems to be a great method for NP-complete problems. Moreover, according to Wood [12], Pardalos and Xue identified in their survey paper [15] the following three key questions that have arisen in a branch-and-bound algorithm for the MCP. We quote from [12]:

1. How to find a good lower bound, i.e., a clique of large size?
2. How to find a good upper bound on the size of maximum clique?
3. How to branch, i.e., break a problem into smaller subproblems?

And so, due to the hard nature of finding even an approximate solution to the MCP quickly, we selected an approach to the problem that combines two separate algorithms, a pruning algorithm, and an enumeration algorithms. The first finds a sequence of improved upper bounds on the maximum clique size, until it finds a clique of size equal to the current bound, and the latter uses backtracking search to build a sequence of larger and larger cliques, and therefore an increasing sequence of lower bounds on the maximum clique size; both find all maximum-size cliques in any given graph.

II. FORMULATING THE MAXIMUM CLIQUE PROBLEM

Thanks to Euler, Graph Theory is thriving.
Year by year it flourishes and blossoms,
Fertilizing much of mathematics
And so rich in all its applications.

Bohdan Zelinka,
The graph theory hymns

A. BOUNDS AND ASSUMPTIONS

In order to develop upper and lower bounds on clique sizes in a given network, and to justify our algorithms in Chapter III, we establish here a few key observations concerning cliques in simple, undirected graphs.

1. Maximum Clique Size Bounds

In [8], it was shown that every graph G with n vertices and minimum vertex degree δ must have a maximum clique of size at least $\left\lceil \frac{n}{n-\delta} \right\rceil$ and that this bound is the best possible in terms of n and δ.

Moreover, we may set a condition for the existence of a maximum clique of size r in an undirected graph :

Observation 1: Consider an arbitrary graph G with n vertices. If G contains a clique of size *r*, then there must be at least *r* vertices, each of which has degree at least $r-1$.

Conversely, if there do not exist at least *r* vertices of degree greater than or equal to *r*-1, then there can be no clique of order *r* in the graph *G*. This condition is used in both of our algorithms to set a quick upper bound for the maximum clique size in a given graph.

2. An Upper Bound on the Number of Cliques of Size *r*

Let G be an undirected graph with n vertices. Then an upper bound on the largest number of cliques of size r (for some r such that $2 \le r \le n$) is the number of subsets of cardinality *r* of the vertex set $V(G)$, namely, $\binom{n}{r}$. It is obvious that this number can be quite big for very large graphs, especially if we search for cliques of size r with $r \approx \frac{n}{2}$. Of course, besides using the subsets of cardinality *r* we also have to count the number of vertices with degree greater than or equal to $r-1$, as was discussed above. And so, in order for $\binom{n}{r}$ cliques of order *r* to exist, G must have at least r vertices of degree greater than or equal to $r-1$. If we denote the number of nodes in *G* of degree at least *r* as n_r, then a better upper bound on the number of cliques of size *r* is $\binom{n_{r-1}}{r}$.

B. GRAPH THEORY AND SOCIAL NETWORK ANALYSIS

As we mentioned above, a variety of the tools and methods widely used in this thesis will come from Graph Theory and Operations Research areas. Moreover, since the applications we will use come from the Social Network Analysis (SNA) area we will also consider a framework to introduce some basic concepts of the latter, and see how all these areas are related to each other. Thus, we will be able to analyze more precisely our results, which arise from the algorithms' implementation on graphs which model social networks, in Chapter V .

1. Graph Theory

A graph is an ordered pair consisting of two sets; the set of vertices, which represent items of interest, and the set of edges, which connect any two distinct vertices that satisfy a particular relation of interest. Hence, a graph $G = (V(G), E(G))$ is uniquely defined by its vertex set $V(G)$ and its edge set $E(G)$, or by a diagram that represents the vertices and edges pictorially. It can also be described by adjacency matrices, incidence matrices or adjacency lists, each of which is useful for particular applications [11, p 48]. In our case, we chose to describe our graphs by their adjacency matrices, which are used as inputs to our algorithms. Other basic characterizations of a graph, or its components, that we use to solve the MCP are *vertex degree*, graph *density*, and *subgraphs*, which are derived by the clique concept and its alternatives. Below we briefly introduce these graph theoretic characteristics we used to solve the MCP.

a. Adjacency Matrix

As we introduced above, a graph $G = (V(G), E(G))$ of *order n* and *size m*, where $V(G) = \{v_1, v_2, ..., v_n\}$ and $E(G) = \{e_1, e_2, ..., e_n\}$ can be described by its adjacency matrix. The *adjacency matrix* of G is the $n \times n$ matrix $A = [a_{ij}]$, where a_{ij}=1 if (v_i,v_j) is in *E*, and is zero otherwise. [11]. We do not admit loops, and so it follows that all the diagonal elements of adjacency matrix must be zeroes: $a_{ii} = 0, \forall i : 1 \le i \le n$.

Moreover, since we consider only undirected graphs, it follows that an edge between a vertex v_i and another one v_j is also an edge between v_j and v_i. In other words, $a_{ij} = a_{ji}, \forall i \ne j$, that is, the adjacency matrix for an undirected graph is a symmetric matrix. The adjacency matrix of a complete graph contains 1s in all off-diagonal cells: $a_{ij} = a_{ji} = 1, \forall i \ne j$.

b. Vertex Degree

The *degree of a vertex v in a graph* G is the number of *edges* incident to *v* and is denoted by $\deg_G v$, or simply $\deg v$ if the graph G is clear from the context [11]. Formally, the degree of each vertex v_i of a graph *G* is $\deg_G v_i = \sum_{j=1}^{n} a_{ij}, \forall i : 1 \le i \le n$ or $\deg_G v_i = |N(v_i)|, \forall i : 1 \le i \le n$, where $N(v_i)$ denotes the set of *neighbors* of, or nodes adjacent to, a vertex $v_i \in V(G)$.

The degree of the vertices of a graph *G* take values that are not arbitrary but are governed under some basic rules and are related to the order and the size of *G*. Hence, we may introduce the First Theorem of Graph Theory and an observation about the limits of degree of the vertices as follows:

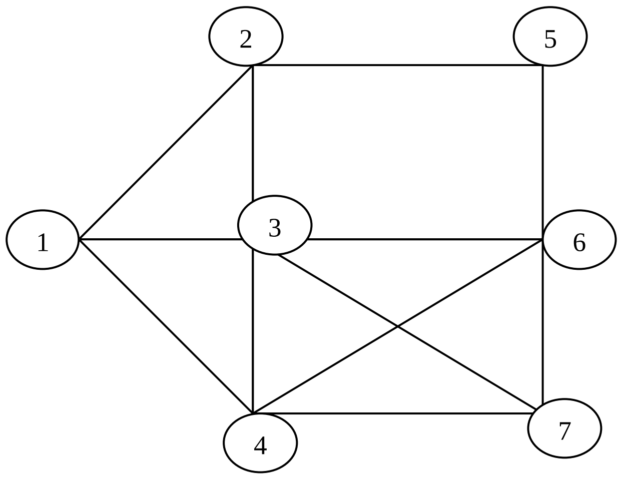
The First Theorem of Graph Theory [11]**:** If G is a graph of size m, then $\sum_{v \in V(G)} \deg_G v = 2m$.

Observation 2: If G is a graph of order n and v is any vertex of G, then: $0 \le \delta(G) \le \deg v \le \Delta(G) \le n-1$, where $\delta(G)$ and $\Delta(G)$ denote the minimum and the maximum degree of *G*, respectively [11].

c. Cliques

We now present the definitions related with the clique and its associated concepts accompanied by several examples. These concepts are strongly related with the MCP and are considered in different kinds of relaxation of the clique problem as we explain in the next section.

Given an undirected graph $G(V,E)$ with *n* vertices and vertex set denoted by $V(G)$, a clique is any complete subgraph of *G*. A clique of order *r* is denoted by K_r. A maximal clique in a *graph* $G(V,E)$ is a *clique* that can not be entirely contained within another *clique* [10], while a maximum clique is the largest complete subgraph of *G*. The order of the maximum clique of *G* is called the clique number of *G*, and is denoted by $\omega(G)$. Examples of nontrivial cliques (a single node or two adjacent nodes form trivial cliques), nontrivial maximal cliques, and the maximum clique of *G* are shown in Figure 1.



Cliques of K_3: {1, 2, 3}, {1, 3, 4}, {3. 4, 6}, {3, 4, 7}, {3, 6, 7}, {4, 6, 7}
Clique of K_4 {3, 4, 6, 7}

Maximal cliques: {1, 2, 3}, {1, 3, 4}, {3, 4, 6, 7}

Maximum clique with $\omega(G) = 4$: {3, 4, 6, 7}

Figure 1. A graph *G* and its nontrivial cliques, maximal cliques and maximum clique. (From [10])

2. Social Network Analysis (SNA)

The pattern of relationships among the members of a group or a larger social system gives the relational structure among them. SNA is used to study this relational structure or any other structural variables measured on actors in the set [10]. According to Memon and Larsen in [19], “SNA in general studies the behavior of the individual at the micro level, the pattern of relationships (network structure) at the macro level, and the interactions between the two.” Moreover, SNA applies techniques to these relationships and investigates how they could be used to infer more information about the actors and groups [20]. More particularly, D. M. Akbar presents the following characteristics that SNA is intended to help identify [21]:

- Important individual, event, place or group.
- Dependency of individual nodes.
- Leader-Follower identification.
- Bonding between nodes.
- Vulnerabilities identification.
- Key players in the network.
- Potential threat from the network.
- Efficiency of overall network.

In SNA, a subgraph in a graph is a clique if it is a maximal complete subgraph of three or more vertices. Mutual dyads are not considered to be cliques. For that reason the restriction that the clique contains at least three nodes is included in the definition [10]. We have to notice that referring to clique here Wasserman and Faust mean maximal clique. Moreover according to Balasundaram et al. in [5], “Clique models idealize three important structural properties that are expected of a cohesive subgroup, namely, familiarity (each vertex has many neighbors and only a few strangers in the group), reachability (a low diameter, facilitating fast communication between the group members) and robustness (high connectivity, making it difficult to destroy the group by removing members).” However, the clique approach has been criticized for its overly restrictive nature [5, 10, 22] and modeling disadvantages [23, 24]. Thus, alternative approaches have been suggested in order to relax the clique definition and different models have been developed in order to relax different aspects of a cohesive subgroup [5]. Hence, we may define the following clique relaxation models according to the property of the cohesive subgroup we want to relax [5,10]:

1.	Complete Mutuality (adjacency):	cliques
2.	Reachability and Diameter (geodesic distance):	k-clique
		k-clans
		k-clubs
3.	Nodal Degree (number of ties among subgroup members):	k-plexes
		k-cores
4.	Relatively Nodal Degree (comparison of frequency of ties within to these ones outside subgroup):	LS Sets
		Lambda Sets

Thus, it is obvious that the MCP we currently consider gives partial information about the structure of social networks. Nevertheless, it may be further considered under the relaxations presented above, in order to provide a more complete idea about the network’s social structure.

III. SOLVING THE MAXIMUM CLIQUE PROBLEM

Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth….

Sherlock Holmes, by Sir Arthur Conan Doyle (1895-1930)

A. INTRODUCTION TO THE PRUNING ALGORITHM

Our *pruning* algorithm is based on the "clique program" developed by Bell [6], which is an algorithm to detect all maximal cliques in a graph. Given a graph *G*, the pruning algorithm uses Observation 1 to establish an upper bound on the maximum clique size, and then searches exhaustively for a clique of that size. If no such clique is found, the upper bound is reduced by one, and the algorithm searches for cliques of that new size. If the upper bound ever becomes two, the algorithm halts (as each edge in the graph is a clique of size two), otherwise, it stops after it has enumerated all cliques of the maximum cardinality in *G*.

Our pruning algorithm is a branch-and-bound algorithm. As we will explain, some of the issues that play a significant role in the Pruning algorithm are (1) the degree of the vertices, (2) the number of vertices of the same degree, (3) the adjacency matrix, as well as (4) its structure, its form and its figure.

According to Observation 1 presented above and considering the degree of the vertices and the number of vertices of the same degree, the algorithm quickly sets an upper bound on the clique number as a first step. Given an undirected graph $G(V,E)$ with n vertices, described by its $n \times n$ adjacency matrix as the input data, the algorithm solves the MCP through the following steps giving the corresponding answers/outputs:

1. Input the $n \times n$ adjacency matrix of G ,
2. Calculate the degree of each vertex,
3. Set an upper bound on the clique number, $\omega(G)$

4. Try to find at least one clique of size $\omega(G)$.

   If such a clique does not exist, reduce $\omega(G)$ and repeat this step (4).

   Otherwise, $\omega(G)$ is the cardinality of a maximum clique in G.

5. Find all subsets of vertices that form maximum cliques for G.

Our contributions beyond Bell's [6] work is a more efficient implementation that is not limited to graphs of very small cardinality (Bell's code was restricted to graphs with up to six vertices), and the development of the last step of the procedure. There, since we have found that there is at least a maximum clique of a particular order, say $\omega(G)$, we find the maximum clique(s) considering only those vertices with degree greater than or equal to $\omega(G)-1$, and using special data structures to help avoid redundant or useless calculations in the enumeration, as presented in the next section.

B. THE PRUNING ALGORITHM

Let $G=\left(V\left(G\right),E\left(G\right)\right)$ be a graph with $n=\left|V\left(G\right)\right|$ vertices and $m=\left|E\left(G\right)\right|$ edges and $v_i \in V\left(G\right)$ any vertex of G. Then $N\left[v_i\right]$ is the set of neighbors of v_i, that is, the set of all adjacent vertices to v_i, together with v_i itself, called the closed neighborhood of v_i. The subgraph induced by $N\left[v_i\right]$ is denoted by $\left\langle N\left[v_i\right]\right\rangle$. The cardinality of $N\left[v_i\right]$ is denoted by $\left|N\left[v_i\right]\right|$; that is, the number of vertices that are adjacent to v_i, together with v_i itself. Similarly, for any $v_i \in V\left(G\right)$, then $N\left(v_i\right)$ is the set of neighbors of v_i, that is, the set of all adjacent vertices to v_i, without v_i itself, called the neighborhood of v_i. The subgraph induced by $N\left(v_i\right)$ is denoted by $\left\langle N\left(v_i\right)\right\rangle$. The cardinality of $N\left(v_i\right)$ is denoted by $\left|N\left(v_i\right)\right|$; that is, the number of vertices that are adjacent to v_i, without v_i itself.

1. Algorithm PRUNING

1. Set the $n \times n$ adjacency matrix $A = \left[a_{ij}\right]$ of G, where a_{ij}=1 if $(i, j) \in E$, and is zero otherwise.

2. $\forall v_i \in V(G)$ find the degree of v_i, $\deg v_i = \sum_{j=1}^{n} a_{ij}, \forall i \in [1, n]$, and calculate the maximum degree in G, $\Delta(G) = \max_i [\deg v_i]$.

3. Set an upper bound on the clique number:

   a. Set $\omega_1(G) = \Delta(G) + 1$.

   b. Let $inducedSum = \sum_{i=1}^{n} b_i$ where $b_i = \begin{cases} 1 & if \deg v_i \geq \omega_1(G) - 1 \\ 0 & otherwise \end{cases}$, $\forall i \in [1, n]$.

   c. If $inducedSum \geq \omega_1(G)$, then go to the next step 4; there might be a maximum clique with clique number $\omega_2(G) = \omega_1(G)$, where $\omega_2(G)$ is again a potential clique number and forms an upper bound for the clique number.

   d. Else if $inducedSum < \omega_1(G)$, then go back to step 3b and set $\omega_1(G) = \omega_1(G) - 1$.

4. Check if there is at least a maximum clique and the clique number $\omega(G)$, $\forall v_i \in V(G) : \deg(v_i) \geq \omega_2(G)$, with $i \in [1, n]$:

   a. Define $N[v_i] : N[v_i] = \{x_1, x_2, ..., x_r\}$, with $1 \leq r \leq \left|N[v_i]\right|$.

   b. Create a vector, say "adjacentVector", where $N[v_i]$ is temporarily stored.

c. Let $inducedSum1 = \sum_{i=1}^{r} c_i$ where $c_i = \begin{cases} 1 & if \deg x_i \geq \omega_2(G)-1 \\ 0 & otherwise \end{cases}$, $\forall i \in [1, r]$.

d. If $inducedSum1 \geq \omega_2(G)$, then go to the next step 5; there is at least one maximum clique of cardinality $\omega(G) \leq \omega_2(G)$.

e. Else if $inducedSum1 < \omega_2(G)$ then go back to step 4 and set $\omega_2(G) = \omega_2(G) - 1$.

5. Find the subset(s) of vertices that form the maximum clique(s) $\forall v_i \in V(G) : \deg(v_i) \geq \omega(G)$, with $i \in [1, n]$:

a. $\forall v_k \in N(v_i)$, with $1 \leq k \leq |N(v_i)|$ define $N(v_k)$: $N(v_k) = \{y_1, y_2, ..., y_s\}$, with $1 \leq s \leq |N(v_k)|$.

b. $\forall v_k \in N(v_i)$, with $1 \leq k \leq |N(v_i)|$ create a vector, say "stack," where $N(v_k)$ is temporarily stored.

c. For $y_1 \in N(v_k)$, that is for the first element of each "stack" execute the following steps:

1) Create a vector, say "stackMaximumClique," where the vertices that form a maximum clique will be stored, that is "*stackMaximumClique*" $= \langle z_1, z_2, ..., z_t \rangle$, with $1 \leq t \leq \omega(G)$.

2) Set $z_1 = v_i, z_2 = v_k, z_3 = y_1$. These three vertices form a K_3 clique so far.

3) Let "*elementA*" $= y_1$ and "*elementB*" $= y_2$. Check if "*elementA*" is adjacent to "*elementB*". (Do this until "*elementA*" $= y_{s-1}$ and "*elementB*" $= y_s$).

a) If yes, then check if “*elementB*” is adjacent to each element of the “stackMaximumClique”:

   i. If yes, then add “*elementB*” in “stackMaximumClique” and go back to step 5.c.3), where set “*elementA*” $= y_2$ and “*elementB*” $= y_3$. Continue this until “*elementA*” $= y_{s-1}$ and “*elementB*” $= y_s$.

   ii. If no, go back to step 5.c.3), where set “*elementB*” $= y_3$. Continue this until “*elementA*” $= y_{s-1}$ and “*elementB*” $= y_s$.

b) If no, then go back to step 5.c.3), where set “*elementB*” $= y_3$. Continue this until “*elementA*” $= y_{s-1}$ and “*elementB*” $= y_s$.

Observation 4: The largest size among the “stackMaximumClique” vectors provides the clique number of the graph.

C. THE ENUMERATION ALGORITHM

Given an undirected graph $G(V,E)$ with n vertices, our *enumeration* algorithm finds all the cliques in *G* through the use of a stack that always contains a list of nodes that comprise a clique of size equal to the number of nodes in the stack, *top*, and successively attempts to add each node not currently in the stack, but adjacent to the top node. If such a node is also adjacent to all other nodes in the stack, it is added to the stack, and the algorithm continues in this manner until no vertices remain to be added. It then backtracks, and attempts to add the next unexplored node to the remaining clique.

1. Algorithm ENUMERATION

```
top = 0
for s = 1 to n begin
    STACK[++top] = s
    onPath[s] = top
    next_node[s] = s+1
    while top > 0 begin
        i = STACK[top]
        while next_node[i]<=n begin
            j=next_node[i]
            while j <= n and adjacent[i,j]==0
                j++
            next_node[i] = j+1
            if (j <= n) begin
                k=1
                while k<top and adjacent[i,STACK[k]]==1
                    k++
                if k==top begin
                    STACK[++top] = j
                    next_node[j] = j+1
                end
            else begin
                top--
            end
            i = STACK[top]
        end
    end
next s
```

2. Discussion

The enumeration algorithm finds all the cliques which exist in an undirected graph and generates each clique exactly once. In order to temporarily store the vertices we have found so far during each turn and form a clique, we use a one-dimensional array with (up to) *n* entries, *STACK*[], that is managed as a stack using the variable *top* as pointer to the top of the stack [25]. Hence, the first *top* entries of *STACK* always contains a list of vertices that form a clique of cardinality *top*. Moreover, considering the vertices in ascending vertex-number order, we start investigate if each one belongs to a clique by setting it as the first element of the "stack." This investigation takes place among this particular vertex and those with higher vertex-numbers. So the vertices which form a clique appear in ascending vertex-number order on the "stack" and, thus, we avoid repetition of cliques.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. APPLICATION OF SOCIAL NETWORK ANALYSIS TO CRIMINAL ACTIVITY: TERRORIST NETWORKS

What social network analysis contributes to counter-terrorism is the ability to map the invisible dynamics inside a terrorist community.

P. V. Fellman and R. Wright [17]

A. THE MODEL

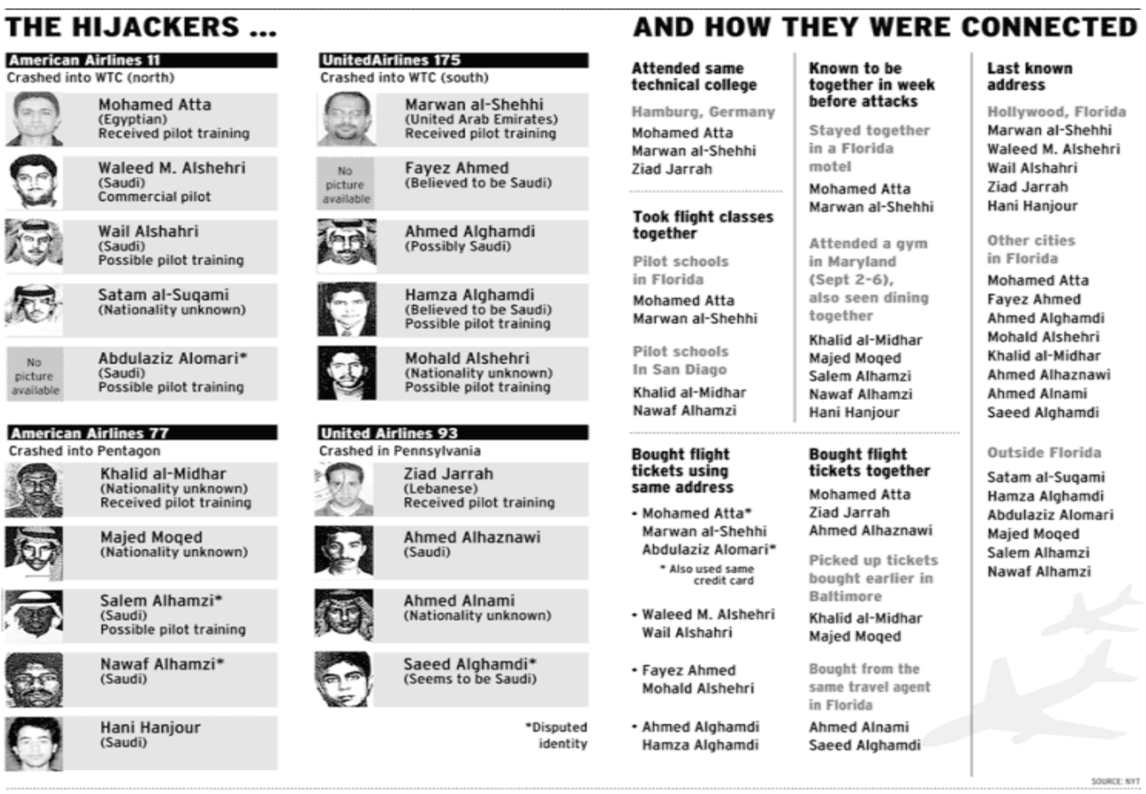
Here we develop a model of a social network to investigate our algorithms' behavior and to examine and qualify their results. More precisely, we want to model and study a real-world situation. So, we focus on terrorist networks, which consist of a direct application area of social networks [5]. According to Balasundaram, et al., in [5], this "is essentially a special application of criminal network analysis that is intended to study organized crimes such as terrorism…."

Throughout the analysis of a particular social network and, more specifically, of a terrorist network, we try to give answers to crucial questions about its structure, like, "who is (are) the leader(s) and how I could identify him(them)?" or, "Are there any active subgroup(s) in the network and how I could recognize it(them)?" The information that we may derive from a network varies, depending on the type of network. According to P. V. Felman et al. in [18], terrorist networks are "first and foremost, covert, which means that they have hidden properties, and our information about them is incomplete." This will become obvious in the next section, where we examine a real-world application of terrorist networks.

B. REAL-WORLD APPLICATION # 1, THE TERRORIST ATTACK OF SEPTEMBER 11, 2001

Hence, as a first application model we selected a terrorist network of an extremely tragic event which marked world history and signaled the start of the GWOT (Global War On Terrorism). This is the terrorist network which depicts the structure and the links among the members who were involved in the terrorist attack of September 11, 2001.

Valdis Krebs used public data that was available before, but collected after the event [5] and constructed a graph. He presented this graph in a remarkable paper, [16], where he made a deep analysis of the terrorist network that caused this terrible attack, based on SNA. He initially mapped a portion of the network centered around the 19 dead hijackers, providing “some insight into the terrorist organization, yet it is incomplete” [16]. Krebs collected information about the 19 hijackers and the relations which connected them, presenting them in a matrix named Early Hijacker Matrix, as is shown below in Figure 2 [16].



THE HIJACKERS ...

American Airlines 11
Crashed into WTC (north)

- Mohamed Atta (Egyptian) Received pilot training
- Waleed M. Alshehri (Saudi) Commercial pilot
- Wail Alshahri (Saudi) Possible pilot training
- Satam al-Suqami (Nationality unknown)
- No picture available Abdulaziz Alomari* (Saudi) Possible pilot training

UnitedAirlines 175
Crashed into WTC (south)

- Marwan al-Shehhi (United Arab Emirates) Received pilot training
- No picture available Fayez Ahmed (Believed to be Saudi)
- Ahmed Alghamdi (Possibly Saudi)
- Hamza Alghamdi (Believed to be Saudi) Possible pilot training
- Mohald Alshehri (Nationality unknown) Possible pilot training

American Airlines 77
Crashed into Pentagon

- Khalid al-Midhar (Nationality unknown) Received pilot training
- Majed Moqed (Nationality unknown)
- Salem Alhamzi* (Saudi) Possible pilot training
- Nawaf Alhamzi* (Saudi)
- Hani Hanjour (Saudi)

United Airlines 93
Crashed in Pennsylvania

- Ziad Jarrah (Lebanese) Received pilot training
- Ahmed Alhaznawi (Saudi)
- Ahmed Alnami (Nationality unknown)
- Saeed Alghamdi* (Seems to be Saudi)

*Disputed identity

AND HOW THEY WERE CONNECTED

Attended same technical college
Hamburg, Germany
Mohamed Atta
Marwan al-Shehhi
Ziad Jarrah

Took flight classes together
Pilot schools in Florida
Mohamed Atta
Marwan al-Shehhi
Pilot schools In San Diago
Khalid al-Midhar
Nawaf Alhamzi

Known to be together in week before attacks
Stayed together in a Florida motel
Mohamed Atta
Marwan al-Shehhi
Attended a gym in Maryland (Sept 2-6), also seen dining together
Khalid al-Midhar
Majed Moqed
Salem Alhamzi
Nawaf Alhamzi
Hani Hanjour

Last known address
Hollywood, Florida
Marwan al-Shehhi
Waleed M. Alshehri
Wail Alshahri
Ziad Jarrah
Hani Hanjour
Other cities in Florida
Mohamed Atta
Fayez Ahmed
Ahmed Alghamdi
Mohald Alshehri
Khalid al-Midhar
Ahmed Alhaznawi
Ahmed Alnami
Saeed Alghamdi
Outside Florida
Satam al-Suqami
Hamza Alghamdi
Abdulaziz Alomari
Majed Moqed
Salem Alhamzi
Nawaf Alhamzi

Bought flight tickets using same address
- Mohamed Atta* Marwan al-Shehhi Abdulaziz Alomari*
  * Also used same credit card
- Waleed M. Alshehri Wail Alshahri
- Fayez Ahmed Mohald Alshehri
- Ahmed Alghamdi Hamza Alghamdi

Bought flight tickets together
Mohamed Atta
Ziad Jarrah
Ahmed Alhaznawi
Picked up tickets bought earlier in Baltimore
Khalid al-Midhar
Majed Moqed
Bought from the same travel agent in Florida
Ahmed Alnami
Saeed Alghamdi

SOURCE: NYT

Figure 2. Early Hijacker Matrix. (From [16])

After completing this matrix he started building a network considering more actors and ties than these ones among the 19 hijackers than he had initially used. Finally, he constructed a graph which mapped this terrorist network, named “Hijacker’s Network Neighborhood” [16] and it is shown below in Figure 3. Moreover, Balasundaram, et al., presented this graph in [5], as shown below in Figure 4, having replaced the names of vertices by numbers. So, for practical reasons, we used this last version of the "Hijacker's Network Neighborhood" as the input into our programs, since each vertex is marked by a number and there is also a correspondence between the names of the hijackers and the numbers of the vertices. This network corresponds to a graph with 37 vertices and 85 edges, that is a graph say $G(37,85)$.

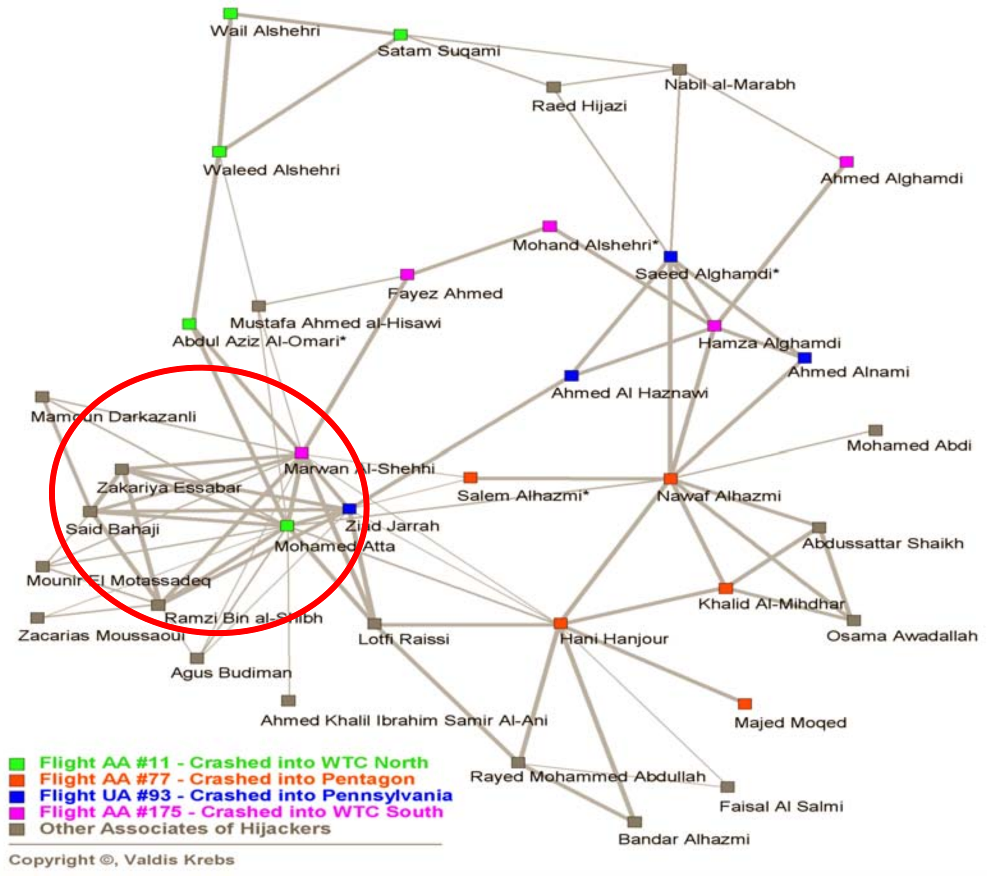


Figure 3. The "Hijacker's Network Neighborhood." (From [16])

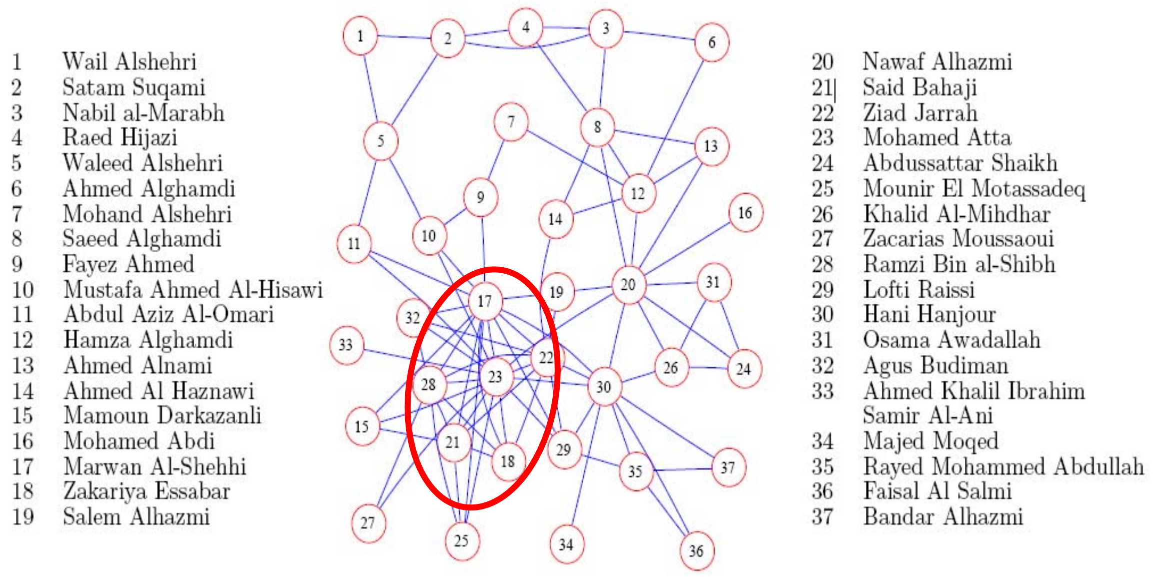


Figure 4. The "Hijacker's Network Neighborhood". (From [5])

After running the two programs, both of them gave us the same correct solution to the MCP. More precisely, they indicated that in the "Hijacker's Network Neighborhood," there is only a maximum clique with clique number six ($\omega(G)=6$). This maximum clique consists of the following vertices 17,18,21,22,23,28 and is marked by a red circle in the graphs of Figure 3 and Figure 4 above. Moreover, the enumeration algorithm provided all the cliques in the graph, as shown on the last pages of Appendix D.

Analyzing the programs' output results, we may present the following comments:

1. First of all it is obvious that our algorithms run correctly and have successfully implemented in the programming language we selected, namely Java.

2. Only three out of the six actors who form the maximum clique were among the 19 hijackers. The other three actors where among their "accomplices who did not get on the planes" [16]. The degrees of the three hijackers of maximum cliques are the largest in the whole network while the three accomplices of maximum clique have not so large degrees. The six actors who form the maximum clique in the graph of "Hijacker's

Network Neighborhood," with their corresponding vertex numbers and their degrees, are shown below, in Table 1.

	Hijackers			**Accomplices**		
Actor's name	Mohammed Atta	Marwan al-Shehhi	Ziad Jarrah	Zakariya Essabar	Said Bahaji	Ramzi Bin alShibh
Vertex number	23	17	22	18	21	28
Degree	15	14	10	5	7	8

Table 1. The actors who consist the maximum clique and their degrees

3. The actors with the highest degree in the graph (greater than or equal to ten) are five, all hijackers, while three of them are the ones who belong to the maximum clique and the other two do not belong to the maximum clique, as is shown below, in Table 2.

	Hijackers belonging to maximum clique			**Hijackers not belonging to maximum clique**	
Actor's name	Mohammed Atta	Marwan al-Shehhi	Ziad Jarrah	Nawaf Alhamzi	Hami Hanjour
Vertex number	23	17	22	20	30
Degree	15	14	10	10	10

Table 2. The actors with the highest degree (greater than or equal to ten)

4. The three hijackers of the maximum clique were pilots of a separate flight, each one as is shown below in Table 3, supposing that they had a leadership role. In this table they are highlighted in yellow. Moreover, two of the hijackers of the fourth flight (Flight 77, which crashed into Pentagon) do not belong to the maximum clique, but they are among the actors with the highest degree in the graph presented above. These last ones are highlighted in red in Table 3. All the others hijackers do not belong to the maximum clique and have intermediate or very small degree in the graph.

American Airlines 11 Crashed into WTC (north)		**United Airlines 11 Crashed into WTC (south)**		**American Airlines 77 Crashed into Pentagon**		**United Airlines 93 Crashed into Pennsylvania**	
Hijacker (Vertex number)	Degree	Hijacker (Vertex number)	Degree	Hijacker (Vertex number)	Degree	Hijacker (Vertex number)	Degree
Mohamed Atta (23)	15	Marwan al-Shehhi (17)	14	Nawaf Alhamzi (20)	10	Ziad Jarrah (22)	10
Waleed M. Alshehri (5)	4	Hamza Alghamdi (12)	6	Hani Hanjour (30)	10	Saeed Alghamdi (8)	6
Satam al-Suqami (2)	4	Ahmed Alghamdi (13)	3	Khalid al-Midhar (26)	4	Ahmed Alhaznawi (14)	3
Abdulaziz Al-Omari (11)	3	Fayez Ahmed (9)	3	Salem Alhamzi (19)	3	Ahmed Alnami (13)	3
Wail. Alshehri (1)	2	Mohald Alshehri (7)	2	Majed Moqed (34)	1		

Table 3. The 19 Hijackers and their Degrees in the "Hijacker's Network Neighborhood"

5. Mohamed Atta seems to play a significant role in this terrorist network since he belongs to the maximum clique and has the highest degree of all 37 actors of the network.

6. The maximum clique cannot give a sufficient and complete answer about the structure and the detection/localization of the leaders and the key actors in the network without considering other parameters or metrics of the graph. If we consider the vertices degree together with the maximum clique, as we analyze above, we may have a deeper understanding of the structure and leadership issues of the graph.

As strong evidence and confirmation of the above results, we quote below some comments/conclusions that Krebs drew in [16], where he did a thorough analysis of the "Hijacker's Network Neighborhood" based on SNA parameters and metrics as the main centrality measures, which are Degrees, Betweenness and Closeness:

- I was amazed at how sparse the network was and how distant many hijackers on the same team where from each other…
- …in the transcript (Department of Defense, 2001) bin Laden mentions:

  > Those who were trained to fly didn't know the others. One group of people did not know the other group….

- Mohamed Atta was the ring leader of this conspiracy…
- Atta scores the highest on Degrees, and Closeness but not Betweenness centrality. These metrics do not necessarily confirm his leader status…

The above analysis is not far from reality and, together with the graph, it would be extremely valuable if the results were developed early enough to prevent this tragic event. Unfortunately, all the related data was collected and connected in a graph after the attack, when the analysis is easier since we know what has already happened.

C. REAL-WORLD APPLICATION # 2, THE TERRORIST ATTACK OF U.S. EMBASSIES IN NAIROBI, KENYA AND DAR ES SALAAM, TANZANIA (AUGUST 7, 1998)

Another application model is also a terrorist network of another tragic event which occurred in two countries of East Africa and the targets were the U.S. Embassies, respectively. On August 7, 1998, two cooperating Al-Qaeda cells carried out bombing attacks against the U.S. Embassies in Nairombi, Kenya and Dar-es-Salaam, Tanzania killing 224 people and wounding over 5000 [26]. The network which represents the ties among the members who were involved in some way in this event is represented by the graph in Figure 5 [27] and is based on data which was collected after the event.

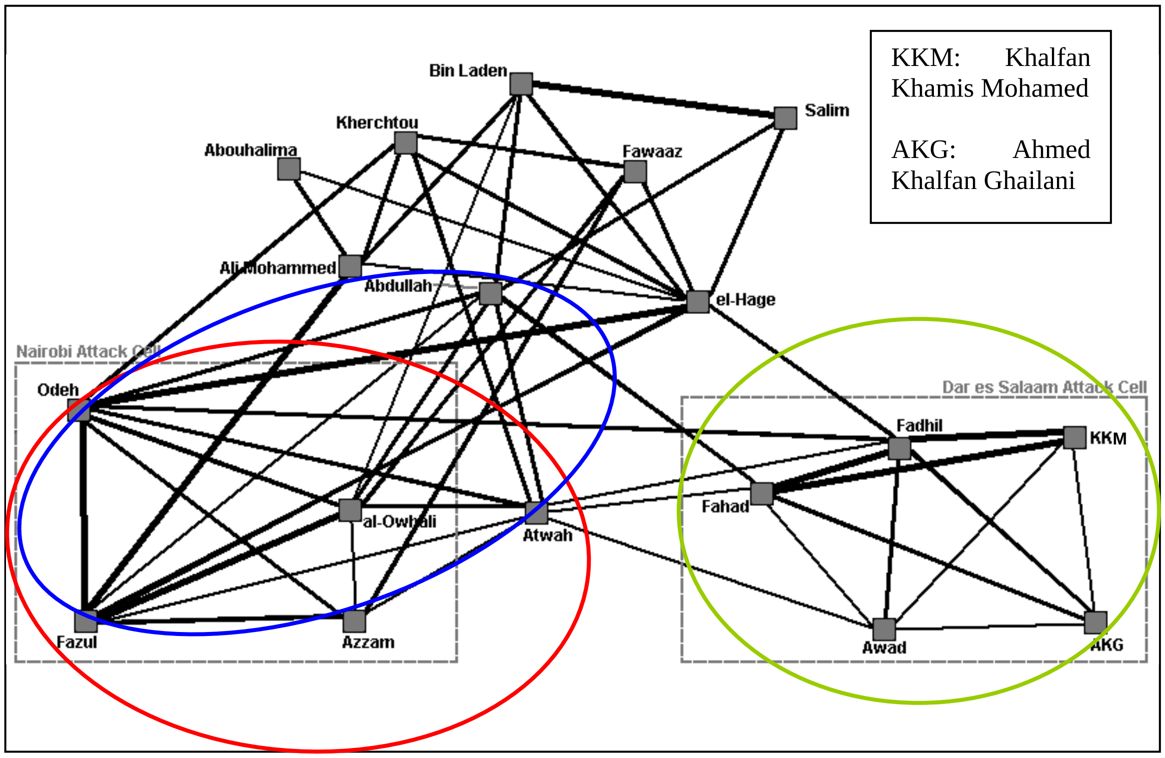


Figure 5. The East Africa U.S. Embassies Attack Network and the corresponding Maximum Cliques. (From [27])

In this case there is a main difference to the group structure comparing with this one of the 9/11 terrorist attack, we presented above. Here, there are two types of cells. The first cell was responsible for the preparation phase while the second ones executed the attacks [26]. This structure is obvious in Figure 5, where the subgroups within the rectangles denotes the two attack cells and the rest actors form the preparation cell. This network corresponds to a graph with 18 vertices and 51 edges, that is a graph say $G(18,51)$.

After running the two programs, both of them gave us the same correct solution to the MCP. More precisely they gave that in the "East Africa U.S. Embassies attack network" there are three maximum cliques with clique number five ($\omega(G)=5$). These maximum cliques are shown in Table 4 and are marked by a different colored circle in the graphs of Figure 5 above.

Max Clique #1		Max Clique #2		Max Clique #3	
Actor's Name	Degree	Actor's Name	Degree	Actor's Name	Degree
Odeh	8	Odeh	8	Fahad	6
Fazul	7	Fazul	7	Fadhil	7
Al-Owhali	7	Al-Owhali	7	Awad	5
Azzam	5	Abdullah	7	KKM	4
Atwah	9	Atwah	9	AKG	4

Table 4. The actors who consist the maximum cliques and their degrees

Analyzing the programs' output results we may present the following comments :

1. First of all it is obvious that our algorithms run correctly and have successfully implemented in the programming language we selected, that is Java.
2. The members of each attack cell form or almost form a maximum clique. The members of the Nairobi attack cell belong to the first two maximum cliques while the five members of Dar-es-Salaam attack cell consist the third maximum clique. Hence, in this case, the concept of maximum cliques depict extremely high valued cohesive subgroups within the whole network, assigned to execute the last and most crucial act of a terrorist attack.
3. The number of actors with the highest degree in the graph (greater than or equal to seven) is seven. In Table 5 we may see their role and its position within the terrorist network, as well as, if they belong to a maximum clique or not. The information related to the actor's role and its position within network, have been retrieved by [26] and [27].

Actor’s Name	**Degree**	**Actor’s Role**	**Actor’s Position**	**Belonging to a Maximum Clique or NOT**
Wahid el-Hage	9	Leader of the East African al-Qaeda cell	Preparation Cell	NOT
Matwalli Atwah	9	He acted as a “communication bridge” among the persons of the two attack cells and the preparation cell.	Preparation Cell	Max. Clique #1 and #2
Mohamed Sadeek Odeh	8	Technical advisor responsible for carrying out the bombings.	Nairobi Attack Cell	Max. Clique #1 and #2
Fazul Abdullah Mohammed	7	Planning/ Orchestration/ Purchaser He was a significant contributor.	Nairobi Attack Cell	Max. Clique #1 and #2
Daoud al-Owhali	7	Suicide Bomber.	Nairobi Attack Cell	Max. Clique #1 and #2
Ahmed Abdullah	7	He replaced el-Hage as the leader of al-Qaeda in East Africa. He was the “mastermind” of the coordinated attack.	Preparation Cell	Max. Clique #2
Mohammed Fadhil	7	Operation Leader.	(Dar-es-Salaam Attack Cell)	Max. Clique #3

Table 5. The actors with the highest degree (greater than or equal to seven)

In the above Table 5, it is obvious that the seven persons with the highest degree within the network played a significant role to the bombing attacks and all except one belong to a maximum clique. The one who does not belong to a maximum clique is the most leading person, named Wahid el Hage, having the highest degree of all the actors. El-Hage had "…the highest number of social interactions…" [27] and acted as the leader of al-Qaeda in East Africa [26] .

4. There were two actors who belong to the first two maximum cliques, without being involved in the Nairobi attack cell, which coincide with the maximum cliques #1 and #2. These ones, named Matwalli Atwah and Ahmed Abdullah, were two key persons who had significant roles in this attack, as it is shown in Table 5.

5. In this case, the maximum cliques give a very good sense about the structure of this network. Together with the number of ties which is the corresponding vertex's degree, they indicate the key persons who played a significant role in this terrorist attack, without considering any other SNA metric.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

Before we fight the enemy we need to 'see' and better understand the enemy...map their networks, figure out their 'patterns' for organizing.

Valdis Krebs

A. CONCLUSIONS-RESEARCH SUMMARY

The main question addressed in this thesis is:

Can we identify and implement an exact algorithm to solve the Maximum Clique Problem (MCP) on undirected graphs, in a reasonable time frame?

We selected two algorithms from the current literature and modified these algorithms to improve their performance, developing the pruning and enumeration algorithms. We then implemented both of these in a modern, powerful and widely used programming language, Java. Our testing involved applying both algorithms to real-world situations which could be modeled by undirected graphs: terrorist social networks.

We verified that both algorithms solve the MCP on undirected graphs, and are quick on relatively small graphs. Furthermore, the pruning algorithm immediately establishes an upper bound on the clique number of a graph and then successively improves this bound; thus the pruning algorithm can be terminated early with a valid upper bound on the clique number. Similarly, the enumeration algorithm quickly finds small cliques, and successively discovers larger and larger cliques in the graph as it progresses, each of which provides a lower bound on the clique number of the graph. It can be terminated early with a valid lower bound on the clique number.

If both are run simultaneously, each provides a bound the other cannot, and an interval of uncertainty can be established that will eventually be reduced to zero, at which point a maximum clique will have been found.

Moreover analyzing a social network using the SNA methods and measures, as we extensively presented in Chapters I and IV, maximum cliques and vertices degree may provide enough information about the social structure of the network under investigation. More particularly, the maximum clique(s) and the clique(s) as well, may give us the most cohesive subgroups, while the vertices with the largest vertex-degrees may show the most central actors in a social network. However, since cliques have been criticized for their restrictive nature as we precisely explain in Chapter I, we may consider and study one of the branches of the relaxation clique problem and analyze a social network under this concept. This may be the subject for an extension of this current thesis and an area for further investigation. It is worthwhile to point out that each algorithm can be modified in fairly straightforward ways to allow various relaxations of the definition of a clique.

B. FUTURE RESEARCH

After presenting the conclusions above, it is obvious that there are many areas and aspects of this problem's approach for improving with further study and future research. Some suggestions are provided below:

1. Since the pruning algorithm generates the maximal cliques with repetitions, it would be more efficient to be modified in such a way to eliminate them and generate the cliques or the maximal cliques or the maximum cliques exactly once.

2. Both computing codes consider as the input data the under investigation graph's adjacency matrix. This method is not so practical for very large graphs, where the user have to set thousands of entries. Hence a modification to the computing code in order to read and set the adjacency matrix from a file text would make it more practical and efficient.

3. Both algorithms' concept to solve the MCP is based on the adjacency matrix of the under investigation graph. Someone may consider another type of data structure to describe the graph and make the algorithms run faster. Such data structure may be as follows:

a. To use a forward-star data structure in addition to the adjacency matrix in order to make finding the next adjacent vertex faster.

b. To use a forward-star data structure where the adjacency list of each vertex would be guaranteed to be sorted in ascending order of tail vertex number.

4. Since cliques have been criticized for their restrictive nature as we precisely explain in Chapter II, someone may consider and study one of the branches of the relaxation clique problem, extend the current algorithms and codes to this direction and analyze a social network under this concept.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. NOTATIONS

$G=(V(G),E(G))$	*graph* consisting of $n=\|V(G)\|$ *vertices* and $m=\|E(G)\|$ *edges.*
V, $V(G)$	*vertex set* of a *graph* G .
E, $E(G)$	*edge set* of a *graph* G .
n	*order of* G , $n=\|V(G)\|$
m	*size of* G , $m=\|E(G)\|$
$\deg_G v$, $\deg v$	*degree* of a *vertex* v in a *graph* G .
$N(v)$	*neighborhood* of a vertex v , $\|N(v)\|=\deg v$
$\langle N(v)\rangle$	subgraph induced by $N(v)$
$N[v]$	*closed neighborhood* of a vertex v
$\langle N[v]\rangle$	subgraph induced by $N[v]$
$H\subseteq G$	H is a *subgraph* of a *graph* G , or G *contains* H *as a subgraph.*
$\langle S\rangle$, $\langle X\rangle$	*induced* or *edge-induced subgraph*
W , $W(u,v)$	$u-v$ *walk* in G
P_n	*path* of order n
C_n	*cycle* of order n
K_n	complete graph of order n
$d_G(u,v)$, $d(u,v)$	*distance* between u and v
$diam(G)$	*diameter* of G

$\overline{G}$	*complement* of a *graph* G
$\overline{K}_n$	*empty graph* of *order* n
$\delta(G)$	*minimum degree* of G
$\Delta(G)$	*maximum degree* of G
$A=\left[a_{ij}\right]$	*adjacency matrix* of G
$B=\left[b_{ij}\right]$	*incidence matrix* of G
$\omega(G)$	*clique number* of G
$w(e)$	weight of an edge e of G
$G_1 \cong G_2$	G_1 is isomorphic to G_2
D	density of G
$C_D(v)$	degree centrality of a vertex v of G
$C_C(v)$	closeness centrality of a vertex v of G
$C_B(v)$	betweenness centrality of a vertex v of G

APPENDIX B. DEFINITIONS RELATED TO GRAPH THEORY

The following terms provide a core working vocabulary for discussing Graph Theory. The definitions for these terms are derived from [11].

Adjacency matrix: Let G be a *graph* of *order* n and *size* m, where $V(G)=\{v_1,v_2,...,v_n\}$ and $E(G)=\{e_1,e_2,...,e_n\}$. The *adjacency matrix* of G is the $n\times n$ matrix $A=\left[a_{ij}\right]$, where $a_{ij}=\begin{cases}1 & \text{if } v_iv_j\in E(G)\\ 0 & \text{otherwise}\end{cases}$.

Adjacent vertices in G **:** two *vertices* are called *adjacent in G* if there is an *edge* between them. In other words if the pair of these *vertices* is an element of the *edge set* $E(G)$.

Bridge: an edge $e=uv$ of a *connected graph* G is called a *bridge* of G if $G-e$ is *disconnected.*

Circuit: a *circuit* in a *graph* is a *closed trail* of *length* 3 or more.

Clique: a *clique* in a *graph* G is a *complete subgraph* of G.

Clique number: the *order* of the *largest clique* in a *graph* G and it is denoted by $\omega(G)$.

Closed neighborhood: the set of *neighbors* of a *vertex* v together with the *vertex* v itself and is denoted by $N[v]$. In other words, $N[v]$=$\mathrm{N}(v)\cup\{v\}$.

Closed walk: a $u-v$ *walk* W in G where $u=v$.

Complement: the *complement* of a *graph* G, denoted by $\overline{G}$, is that *graph* whose *vertex set* is $V(G)$ and such that for each pair *u, v* of *vertices* of G, uv is an *edge* of $\overline{G}$ if and only if uv is not an *edge* of G. Observe that if G is a *graph* of *order* n and *size* m, then $\overline{G}$ is a *graph* of *order* n and *size* $\binom{n}{2}-m$.

Complete: a graph G is complete if every two distinct vertices of G are adjacent. A complete graph of order n is denoted by K_n. Therefore, K_n has the maximum possible *size* for a *graph* with n *vertices***.**

Connected: let u and v be two *vertices* of a *graph* G. Then u *is connected to* v (and also v *is connected to* u) if G contains a $u-v$ *path in* G. So, saying that u and v are connected only means that there is some $u-v$ *path* in G; it doesn't say that u and v *are necessarily joint by an edge*.

Connected graph: a *graph* G is *connected* if every two *vertices* of G are connected.

Cut-vertex: a vertex v in a *connected graph* G is a *cut-vertex* of G if $G-v$ is *disconnected.*

Cycle: is a *circuit* that repeats no *vertex*, except for the first and last. A *k-cycle* is a *cycle* of *length k.*

Degree: the *degree of a vertex v in* G is the number of *edges* incident to *v* and is denoted by $\deg_G v$, or simply $\deg v$ if the graph G is clear from the context.

Degree sequence: if the degrees of vertices of a graph G are listed in a sequence *s***,** then *s* is called a degree sequence of G**.**

Density of a graph: it is the proportion of possible *edges* that are actually present in the *graph*. It is the ratio of number of *edges* present to the maximum possible and is denoted by D.

Diameter: the greatest *distance* between any two *vertices* of a *connected graph* G is called the *diameter* of G and is denoted by $diam(G)$.

Digraph (or **Directed graph):** is a finite nonempty set V of objects called *vertices* together with a set E of ordered pairs of distinct *vertices*.

Disconnected graph: a *graph* G is called *disconnected* if it is not connected.

Distance: the distance between u and v is the smallest *length* of a $u-v$ *path* in G and is denoted by $d_G(u,v)$ or simply by $d(u,v)$ if the graph G under consideration is clear. Hence if $d(u,v)=k$, then there exists a $u-v$ *path* $P: u=v_0, v_1, ..., v_k = v$, of length k in G, but no $u-v$ *path* of smaller length exists in G.

Edge: is a 2-element subset of the *vertex set* $V(G)$. *Edges* are sometimes called *lines*.

Edge-induced subgraph of G: is the *subgraph* $\langle X \rangle$ *induced by* X which has *edge set* X *and* consists of all *vertices* that are *incident* with at least one *edge* in X

Edge set: is the set $E(G)$ consisting of all the *edges* of a *graph G*.

Empty graph: the graph that has n vertices and no edges is called the empty graph of order n and is denoted by $\overline{K}_n$.

Geodesic path: a $u-v$ *path* of *length* $d(u,v)$ is called a $u-v$ *geodesic*.

Graph: a *graph* $G=(V(G),E(G))$ is an ordered pair of two sets $V(G)$ and $E(G)$, where V is a finite nonempty set of objects called *vertices* (the singular is *vertex*) and E is a set of 2-element subsets of V called *edges*. One could also use $G=(V,E)$. At times, it is useful to write $V(G)$ and $E(G)$ rather than V and E to emphasize that these are the *vertex* and *edge sets* of a particular *graph* G. The *graph* $G=(V(G),E(G))$ is consisting of $n=|V(G)|$ *vertices* and $m=|E(G)|$ *edges*.

Incidence matrix: Let G be a *graph* of *order n* and *size m*, where $V(G)=\{v_1,v_2,...,v_n\}$ and $E(G)=\{e_1,e_2,...,e_m\}$. The *incidence matrix* of G is the $n\times m$ matrix $B=[b_{ij}]$, where $b_{ij}=\begin{cases}1 & \text{if } v_i \text{ is incident with } e_j\\ 0 & \text{otherwise}\end{cases}$.

Induced subgraph: a *subgraph* F of a *graph* G is called *an induced subgraph* of G if whenever u and v are vertices of F and uv is an edge of G, then uv is an edge of F as well.

Isomorphism: it is the one-to-one correspondence ϕ from $V(G_1)$ to $V(G_2)$ such that $u_1v_1 \in E(G_1)$ if and only if $\varphi(u_1)\varphi(v_1) \in E(G_2)$.

Isomorphic graphs: two *graphs* G_1 and G_2 are *isomorphic* (have the same structure) if there exists a one-to-one correspondence ϕ from $V(G_1)$ to $V(G_2)$ such that $u_1v_1 \in E(G_1)$ if and only if $\varphi(u_1)\varphi(v_1) \in E(G_2)$. If G_1 and G_2 are *isomorphic graphs*, then we say that G_1 is *isomorphic to* G_2 and we write $G_1 \cong G_2$.

Length of a path: is the number of *edges* encountered in a path.

Length of a walk: is the number of *edges* encountered in a *walk* (including multiple occurrences of an *edge if used in the walk*).

Maximal: a *subgraph* is said to be *maximal* with respect to some property if that property holds for the subgraph, but does not hold if additional *vertices* and the *edges incident* with them are added to the *subgraph* [10].

Maximal clique: a *maximal clique* in a *graph* G is a *clique* that can not be entirely contained within another *clique* [10].

Maximum clique: a *maximum clique* in a *graph* G is the largest *complete subgraph* of G.

Maximum degree: the *maximum degree* of G is the maximum *degree* among the *vertices* of G. It is denoted by $\Delta(G)$.

Minimum degree: the *minimum degree* of G is the minimum *degree* among the *vertices* of G. It is denoted by $\delta(G)$.

Multigraph: consists of a finite nonempty set V of vertices and a set E of edges, where every two vertices are joined by a finite number of edges (possibly zero)

Neighborhood of v: is the set $N(v)$ of *neighbors* of a vertex v. The *neighborhood of v* is sometimes called *open neighborhood of v*. The cardinality of the *neighborhood of v* equals to the *degree of v*, that is $\deg v = |N(v)|$.

Neighbors: two *vertices* are called *neighbors*, if they are *adjacent in G*.

Non-isomorphic graphs: if two graphs G_1 and G_2 are not *isomorphic*, then they are called *non-isomorphic graphs*.

Non-regular graph: a *graph* G ,which at least two *vertices* have not the same *degree*, is called *non-regular*.

Nontrivial graph: is a *graph* with *order* at least 2.

Open walk: is a $u-v$ walk W in G where $u \neq v$.

Order of G : is the number of *vertices* in G , in other words the cardinality of the *vertex set* $V(G)$. That is the *order of G* is $n = |V(G)|$.

Path: a $u-v$ *path* W in G is a $u-v$ walk in which no vertices is repeated. If no *vertex* in a *walk* is repeated, then no *edge* is repeated either. Hence every *path* is a *trail*, while not every *trail* is a *path*.

Proper subgraph: a *graph* H is called a *proper subgraph* of a *graph* G if $H \subseteq G$ and either $V(H) \subset V(G)$ or $E(H) \subset E(G)$.

Regular: a *graph* G , where $\delta(G) = \Delta(G)$, that is all the *vertices* of G have the same *degree,* is called *regular*. If $\deg v = r$ for every *vertex* v of G , where $0 \leq r \leq n-1$, then G is *r-regular or regular of degree r*.

Size of G : is the number of *edges* in G , or the cardinality of the *edge set* $V(G)$. That is the *size of G* is $m = |E(G)|$.

Spanning subgraph: if a *subgraph* H of a *graph* G has the same vertex set as G , that is $V(H) = V(G)$, then H is *a spanning subgraph* of G .

Subgraph: a *graph* H is called a *subgraph* of a *graph* G, written $H \subseteq G$ **,** if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We also say that G *contains* H as a *subgraph.*

Subgraph of G **induced by** S **:** if S is a nonempty set of *vertices* of a *graph* G, then the *subgraph of* G *induced by* S is the *induced subgraph* with *vertex set* S. It is denoted by $\langle S \rangle$ or by $\langle S \rangle_G$ to emphasize that this is an *induced subgraph* of G

Trail: a $u-v$ *trail* W in G is a $u-v$ walk in which no *edge* is traversed more than once, that is no *edge* is repeated, while no such condition is placed on *vertices*.

Trivial graph: is a *graph* with exactly one *vertex*.

Trivial walk: is a *walk* of *length* 0.

Unweighted graph: a *graph* G each of whose *edges* is not assigned a number (called *cost* or *weight* of the edge) forms an un*weighted graph*.

Vertex (Vertices): is a combinatorial element in terms of which a *graph* is defined. *Vertices* are sometimes called *points* or *nodes*.

Vertex set: is the set $V(G)$ consisting of all the *vertices* of a *graph* G .

Walk: a $u-v$ walk W in G is a sequence of *vertices* in G, beginning with u and ending at v such that consecutive vertices in the sequence are adjacent. That is to say that W can be expressed as W: $u = v_0, v_1, ..., v_k = v$, where $k \geq 0$ and v_i and v_{i+1} are adjacent for $i = 0,1,2,...,k-1$. Each vertex v_i $(0 \leq i \leq k)$ and each edge $v_i v_{i+1}$ $(0 \leq i \leq k-1)$ is said to *lie on, or belong to* W .

Weighted graph: a *graph* G each of whose *edges* is assigned a number (called *cost* or *weight* of the *edge*) forms a *weighted graph*. The *weight* of an edge e of G is denoted by $w(e)$.

APPENDIX C. SOCIAL NETWORK ANALYSIS DEFINITIONS

The following terms provide a core working vocabulary for discussing Social Network Analysis. The definitions for these terms are derived by [10].

Actor: it is the social entity. *Actors* are discrete individual, corporate, or collective social units.

Actors set: it is the entire collection of *actors* on which we take measurements.

Affiliation Network: it is a *two-mode social network*, in which only one set has *actors*, while the second *mode* is a set of *events* to which the *actors* belongs.

Centrality: the centrality determines the importance of an actor within the network. It is denoted by $C_A(v_i)$, where $v_i \in V(G)$ and A is a generic measure.

Cohesive subgroup: a cohesive subgroup is a subset of actors among whom there are relatively strong, direct, intense, frequent or positive ties.

Dichotomous relations: they are those binary *relations* that are coded as either present or absent for each pair of *actors*.

Directional relation: it is the *relation* where the *relational tie* between a pair of *actors* has an origin and a destination; that is the tie is directed from one *actor* in a pair to the other *actor* in the pair.

Dyad: a *dyad* consists of a pair of *actors* and the (possible) *tie(s)* between them. It is the unit of *social network analysis*.

Event: the *events* are often defined on the basis of membership in clubs or voluntary organizations, attendance at social events, sitting on a board of directors or socializing in a small group.

Group: a group is a collection of all *actors* on which *ties* are to be measured.

Mode: a mode is a distinct set of entities on which the structural variables are measured.

Multiple Relations: they are the case where more than one relations are measured on a single set of actors.

Non-directional relation: it is the tie between a pair of *actors* which does not have a direction.

One-mode network: it is a *social network* where all *actors* are all of the same type, i.e. people in a work group. It is the most common type of network, since all *actors* come from one set.

Prestige: it is a property that characterizes an actor in a directed network with directional relations and qualify an actor by the number of indegrees ties rather than the outdegrees ones.

Relation: it is the collection of ties of specific kind among members of a *group*.
Relational tie: a *tie* establishes a linkage between a pair of *actors*. It is also called *social tie*.

Single Relation: it is a *relation* where each *actor* in the *actor set* relates to every other *actor* of this *relation*.

Social Network: a *social network* consists of a finite set(s) of *actors* and the *relations* defined on them.

Social Network data: consists of one (or more) *relations* measured among a set of *actors*.

Subgroup: a *subgroup of actors* is any subset of *actors* and all (possible) *ties* among them.

Triad: it is a subset of three *actors* and the (possible) *tie(s)* among them.

Valued relations: they are those *relations* which can take a range of values, indicating the strength, intensity, or frequency of the ties between each pair of *actors*.

APPENDIX D. THE CODE OF THE ENUMERATION ALGORITHM

```
public class All_Cliques_Enum_Adj_Terror_Thesis {
      /*
       * File: Maximum Clique
       *
       * Created on April 03, 2008
       *
       * 1st Modified on April 06, 2008
       * 2nd Modified on April 28, 2008
       *
       * ************THIS WORKS FOR the Terror Graph G(37,85) WITH A K6
MAX CLIQUE, BASED ON ENUMARATION METHOD *****
       *
       *
       */

      public static void main(String[] args) {

            // initialize instance variables
            int vectorsNumber = 37;
            int[] degree = new int[vectorsNumber];
            int[] booleanDegree = new int[vectorsNumber];
            int maxCliqueDegree = 0;
            int tempMaxCliqueDegree = 0;// This is equal to (Potential
Maximum Clique # -1), (PMC#-1). Its value decreases upon the condition
which are met, until Maximum Clique is found. Then Clique#=(updated by
the algorithm)PMC#
            int initialTempMaxCliqueDegree = 0;// This is equal to the
initial value of the (Potential Maximum Clique # -1), (PMC#-1). Its
value remains constant.
            int newTempMaxCliqueDegree = 0;
            boolean maxClique = false;
            boolean initialMaxClique = false;
            int cliqueNumber = -1;// this is the clique number
            int[][] table = new int[vectorsNumber][vectorsNumber];//
This is the nxn adjacency matrix of the given graph G (with n vertices).

            int[] count = new int[vectorsNumber];// counter how many
Maximum Cliques there exist
            int[][] maximumClique = new int[40][vectorsNumber];// This
            int[][][] clique = new
int[vectorsNumber][1140][vectorsNumber];
            int row, column;
            //
************************************************************************
****************************************************
            // I. PUT AN UPPER BOUND

            // 1. Set the nxn adjacency matrix of the given graph G
(with n vertices).
            table[0][0] = 0;
            table[0][1] = 1;
            table[0][2] = 0;
            table[0][3] = 0;
            table[0][4] = 1;
            .............................
            .............................
            table[36][33] = 0;
            table[36][34] = 1;
            table[36][35] = 0;
            table[36][36] = 0;

            // ****************

            System.out.println("Maximum Clique Problem of G(" +
vectorsNumber + ",85)");
            System.out.println();

            // 2. Find the degree of each vertex by adding the entries
of each row of the adjacency matrix.
            System.out.println("Find the degree of each vertex ");
            for (row = 0; row < vectorsNumber; row++) {
                  int sum = 0;// This is the degree of each vertex
                  for (column = 0; column < vectorsNumber; column++)
                        sum = sum + table[row][column];
                  degree[row] = sum;
            }
            // print the degree of each vertex
            for (row = 0; row < vectorsNumber; row++) {
                  System.out.println("degree(v" + (row + 1) + " )= " +
degree[row]);
            }
            System.out.println();

            // 3. Find the Potential Max Clique#, say PMQ#,
            // (Potential Max Clique# = max Vertex Degree+1).
            for (row = 0; row < vectorsNumber; row++)
                  if (degree[row] > tempMaxCliqueDegree) {
                        tempMaxCliqueDegree = degree[row];
                  }
            System.out.println("Potential Max Clique Degree: "
                        + (tempMaxCliqueDegree + 1) + ".");

            initialTempMaxCliqueDegree = tempMaxCliqueDegree;

            // 4. Let v_k the vertices with degree greater than or equal
to (PMC#-1),for some k, 1<=k<=n. Find the # of vertices v_k with degree
greater than or equal to (PMC#-1), say “inducedSum”.
            do {
                  int inducedSum = 0;// the # of vertices with degree
greater than or equal to (PMC#-1)
                  for (row = 0; row < vectorsNumber; row++) {
                        if (degree[row] >= tempMaxCliqueDegree) {
                              booleanDegree[row] = 1;
                              inducedSum = inducedSum +
booleanDegree[row];
                        } else
```

```
                                booleanDegree[row] = 0;
                        }

                        // 4a. If “inducedSum” is greater than or equal to
PMC#, then maybe there is a Max Clique with Clique# = PMC#.
                        if (inducedSum >= tempMaxCliqueDegree + 1) {
                              System.out.println("Maybe there is a K"
                                          + (tempMaxCliqueDegree + 1) + " Max
Clique.");
                              System.out.println();
                              initialMaxClique = true;

                              // 4b. Else if “inducedSum” is NOT greater than
or equal to PMC#, there is not a Maximum Clique with Clique# = PMQ#. Set
PMQ#=PMQ#-1 and continue by going back to step 4.
                        } else {
                              System.out.println("There is not a K"
                              + (tempMaxCliqueDegree + 1) + " Max Clique.");
                              System.out.println("Check for K"
                              + (tempMaxCliqueDegree) + " Max Clique.");
                              System.out.println();
                              tempMaxCliqueDegree--;
                        }
                  } while (tempMaxCliqueDegree > 0 && initialMaxClique ==
false);
                  //
************************************************************************
*********************************************
int top = -1;
                  for (row = 0; row < vectorsNumber; row++) {

                        int[] stack = new int[vectorsNumber];// This
                        // is the vector where the vertices of the
                        // neighborhood of each element of N[V_k] are
                        // stored.

                        int[] nextNode = new int[vectorsNumber];// This
                        // is the vector where the vertices consisting a
                        // Maximum Clique are stored.

                        stack[++top] = row;
                        nextNode[row] = row + 1;

                        while (top >= 0) {
                              int i = stack[top];

                              while (nextNode[i] <= vectorsNumber) {
                                    int j = nextNode[i];

                                    while (j < vectorsNumber && table[i][j] ==
0) {
                                          j++;
                                    }
                                    nextNode[i] = j + 1;
                                    if (j < vectorsNumber) {
                                          int k = 0;
                                          while (k < top && table[j][stack[k]]
      == 1) {
                                                k++;
                                          }
                                          if (k == top) {
                                                top++;
                                                stack[top] = j;
                                                nextNode[j] = j + 1;
                                                System.out.println();
                                                System.out.println();

                                                if (top > cliqueNumber) {
                                                      cliqueNumber = top;
                                                }
                                                for (int spyros = 1; spyros <=
      tempMaxCliqueDegree; spyros++) {
                                                      if (top == spyros) {


            System.out.println();

            System.out.println("There is a K"   + (top + 1) + " Clique.");


            System.out.println("The Clique is: ");
                                          for (int m = 0; m <= top; m++) {

            System.out.print((stack[m] + 1) + " ");
                        clique[top + 1][count[top + 1]][m] = stack[m];
                                                            }

            System.out.println();
                                                            count[top + 1]++;
                                                      }

                                                }
                                          }
                                    } else {
                                          top--;
                                    }
                                    if (j < vectorsNumber) {
                                          i = stack[top];
                                    }
                              }
                        }
                  }
                  int[] maximumClique1 = new int[cliqueNumber + 1];
                  // print the outputs
                  System.out.println();
                  System.out.println("The Maximum Clique is K" + (cliqueNumber
      + 1));
                  System.out.println("There is " + count[cliqueNumber + 1]
                              + " Maximum Clique");
```

```
            System.out.println("The Maximum Clique is: ");
            // for (int k = 0; k <vectorsNumber; k++) {
            for (int n = 0; n < count[cliqueNumber + 1]; n++) {
                  System.out.print((n + 1) + ") ");
                  for (int m = 0; m < cliqueNumber + 1; m++) {
                        System.out.print((clique[cliqueNumber + 1][n][m]
+ 1) + " ");
                        maximumClique1[m] = clique[cliqueNumber +
1][n][m] + 1;
                                          }
                  System.out.println();
            }
            System.out.println();
            //
*************************************************************************
***

            //
*************************************************************************
******
            for (int k = 0; k < cliqueNumber; k++) {
                  System.out.println("There are " + count[(cliqueNumber
+ 1) - k]
                              + " Cliques K" + ((cliqueNumber + 1) -
k));
                  System.out.println("The Cliques are: ");
                  // for (int k = 0; k <vectorsNumber; k++) {
                  for (int n = 0; n < count[(cliqueNumber + 1) - k];
n++) {
                        System.out.print((n + 1) + ") ");
                        for (int m = 0; m < (cliqueNumber + 1) - k; m++)
{
                              System.out.print((clique[(cliqueNumber +
1) - k][n][m] + 1)
                                          + " ");
                              // System.out.print((clique[k][n][m]+1) +
" ");
                        }
                        System.out.println();
                  }
                  System.out.println();
            }

      }
}
```

THE OUTPUT

```
Maximum Clique Problem of G(37,85)

Find the degree of each vertex
degree(v1 )= 2
degree(v2 )= 4
degree(v3 )= 4
degree(v4 )= 3
degree(v5 )= 4
degree(v6 )= 2
degree(v7 )= 2
degree(v8 )= 6
degree(v9 )= 3
degree(v10 )= 4
degree(v11 )= 3
degree(v12 )= 6
degree(v13 )= 3
degree(v14 )= 3
degree(v15 )= 3
degree(v16 )= 1
degree(v17 )= 14
degree(v18 )= 5
degree(v19 )= 3
degree(v20 )= 10
degree(v21 )= 7
degree(v22 )= 10
degree(v23 )= 15
degree(v24 )= 3
degree(v25 )= 4
degree(v26 )= 4
degree(v27 )= 2
degree(v28 )= 8
degree(v29 )= 5
degree(v30 )= 10
degree(v31 )= 3
degree(v32 )= 4
degree(v33 )= 1
degree(v34 )= 1
degree(v35 )= 4
degree(v36 )= 2
degree(v37 )= 2

Potential Max Clique Degree: 16.
There is not a K16 Max Clique.
Check for K15 Max Clique.

There is not a K15 Max Clique.
Check for K14 Max Clique.

There is not a K14 Max Clique.
Check for K13 Max Clique.

There is not a K13 Max Clique.
Check for K12 Max Clique.
```

```
There is not a K12 Max Clique.
Check for K11 Max Clique.

There is not a K11 Max Clique.
Check for K10 Max Clique.

There is not a K10 Max Clique.
Check for K9 Max Clique.

There is not a K9 Max Clique.
Check for K8 Max Clique.

There is not a K8 Max Clique.
Check for K7 Max Clique.

Maybe there is a K7 Max Clique.

There is a K2 Clique.
The Clique is:
1 2

There is a K3 Clique.
The Clique is:
1 2 5
……………………………………………………………
……………………………………………………………
The Maximum Clique is K6
There is 1 Maximum Clique
The Maximum Clique is:
1) 17 18 21 22 23 28

There is 1 Clique K6
The Cliques are:
1) 17 18 21 22 23 28

There are 9 Cliques K5
The Cliques are:
1) 17 18 21 22 23
2) 17 18 21 22 28
3) 17 18 21 23 28
4) 17 18 22 23 28
5) 17 21 22 23 28
6) 17 21 23 25 28
7) 17 22 23 28 32
8) 17 22 23 29 30
9) 18 21 22 23 28

There are 31 Cliques K4
The Cliques are:
1) 8 12 13 20
2) 15 17 21 23
3) 17 18 21 22
4) 17 18 21 23
5) 17 18 21 28
```

```
6) 17 18 22 23
7) 17 18 22 28
8) 17 18 23 28
9) 17 21 22 23
10) 17 21 22 28
11) 17 21 23 25
12) 17 21 23 28
13) 17 21 25 28
14) 17 22 23 28
15) 17 22 23 29
16) 17 22 23 30
17) 17 22 23 32
18) 17 22 28 32
19) 17 22 29 30
20) 17 23 25 28
21) 17 23 28 32
22) 17 23 29 30
23) 18 21 22 23
24) 18 21 22 28
25) 18 21 23 28
26) 18 22 23 28
27) 20 24 26 31
28) 21 22 23 28
29) 21 23 25 28
30) 22 23 28 32
31) 22 23 29 30

There are 66 Cliques K3
The Cliques are:
1) 1 2 5
2) 2 3 4
3) 3 4 8
4) 8 12 13
5) 8 12 14
6) 8 12 20
7) 8 13 20
8) 9 10 17
9) 10 17 23
10) 11 17 23
11) 12 13 20
12) 15 17 21
13) 15 17 23
14) 15 21 23
15) 17 18 21
16) 17 18 22
17) 17 18 23
18) 17 18 28
19) 17 19 22
20) 17 21 22
21) 17 21 23
22) 17 21 25
23) 17 21 28
24) 17 22 23
25) 17 22 28
26) 17 22 29
```

```
27) 17 22 30
28) 17 22 32
29) 17 23 25
30) 17 23 28
31) 17 23 29
32) 17 23 30
33) 17 23 32
34) 17 25 28
35) 17 28 32
36) 17 29 30
37) 18 21 22
38) 18 21 23
39) 18 21 28
40) 18 22 23
41) 18 22 28
42) 18 23 28
43) 20 23 30
44) 20 24 26
45) 20 24 31
46) 20 26 30
47) 20 26 31
48) 21 22 23
49) 21 22 28
50) 21 23 25
51) 21 23 28
52) 21 25 28
53) 22 23 28
54) 22 23 29
55) 22 23 30
56) 22 23 32
57) 22 28 32
58) 22 29 30
59) 23 25 28
60) 23 27 28
61) 23 28 32
62) 23 29 30
63) 24 26 31
64) 29 30 35
65) 30 35 36
66) 30 35 37

There are 85 Cliques K2
The Cliques are:
1) 1 2
2) 1 5
3) 2 3
4) 2 4
5) 2 5
6) 3 4
7) 3 6
8) 3 8
9) 4 8
10) 5 10
11) 5 11
12) 6 12
```

```
13) 7 9
14) 7 12
15) 8 12
16) 8 13
17) 8 14
18) 8 20
19) 9 10
20) 9 17
21) 10 17
22) 10 23
23) 11 17
24) 11 23
25) 12 13
26) 12 14
27) 12 20
28) 13 20
29) 14 22
30) 15 17
31) 15 21
32) 15 23
33) 16 20
34) 17 18
35) 17 19
36) 17 21
37) 17 22
38) 17 23
39) 17 25
40) 17 28
41) 17 29
42) 17 30
43) 17 32
44) 18 21
45) 18 22
46) 18 23
47) 18 28
48) 19 20
49) 19 22
50) 20 23
51) 20 24
52) 20 26
53) 20 30
54) 20 31
55) 21 22
56) 21 23
57) 21 25
58) 21 28
59) 22 23
60) 22 28
61) 22 29
62) 22 30
63) 22 32
64) 23 25
65) 23 27
66) 23 28
67) 23 29
```

```
68) 23 30
69) 23 32
70) 23 33
71) 24 26
72) 24 31
73) 25 28
74) 26 30
75) 26 31
76) 27 28
77) 28 32
78) 29 30
79) 29 35
80) 30 34
81) 30 35
82) 30 36
83) 30 37
84) 35 36
85) 35 37
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. THE CODE OF THE PRUNING ALGORITHM

```
/*      * File: Maximum Clique
        *
        * Created on March 03, 2008
        *
        * Modified on March 012, 2008
        *
        *
        * ***********THIS WORKS FOR the GRAPH "911", G(37,85) where there
is a K6 (17,18,21,22,23,28)
        *
        *
        *  Let  G=(V(G),E(G))  be  a  graph  with  n=|V(G)|  vertices  and
m=|E(G)| edges.
        *  Let v_i belongs to V(G). Then N[V_i]=set of neighbors of v_i,
together with v_i itself, called the closed neighborhood. The subgraph
induced by* N[V_i] is denoted by <N[V_i]>.
        */

      public static void main(String[] args) {

            // initialize instance variables
            int vectorsNumber = 37;
            int[] degree = new int[vectorsNumber];
            int[] booleanDegree = new int[vectorsNumber];
            int maxCliqueDegree = 0;
            int tempMaxCliqueDegree = 0;// This is equal to (Potential
Maximum Clique # -1), (PMC#-1). Its value decreases upon the condition
which are met, until Maximum Clique is found. Then Clique#=(updated by
the algorithm)PMC#
            int initialTempMaxCliqueDegree = 0;// This is equal to the
initial value of the (Potential Maximum Clique # -1), (PMC#-1). Its
value remains constant.
            int newTempMaxCliqueDegree = 0;
            boolean maxClique = false;
            boolean initialMaxClique = false;
            int[][]  table  =  new  int[vectorsNumber][vectorsNumber];//
This is the nxn adjacency matrix of the given graph G (with n vertices).
            int[][] maximalClique = new int[1000][vectorsNumber];
            int countMaximalCliques=0;
            int row, column;
//*********************************************************************
******************************************************
            // I. PUT AN UPPER BOUND

            // 1. Set the nxn adjacency matrix of the given graph G
(with n vertices).
            table[0][0] = 0;
            table[0][1] = 1;
            table[0][2] = 0;
....................................
....................................
```

```
            table[36][34] = 1;
            table[36][35] = 0;
            table[36][36] = 0;

            // ****************

            System.out.println("Maximum Clique Problem of G(37,85)");
            System.out.println();

            // 2. Find the degree of each vertex by adding the entries
of each row of the adjacency matrix.
            System.out.println("Find the degree of each vertex ");
            for (row = 0; row < vectorsNumber; row++) {
                  int sum = 0;// This is the degree of each vertex
                  for (column = 0; column < vectorsNumber; column++)
                        sum = sum + table[row][column];
                  degree[row] = sum;
            }
            // print the degree of each vertex
            for (row = 0; row < vectorsNumber; row++) {
                  System.out.println("degree(v" + (row + 1) + " )= " +
degree[row]);
            }
            System.out.println();

            // 3. Find the Potential Max Clique#, say PMQ#,
            // (Potential Max Clique# = max Vertex Degree+1).
            for (row = 0; row < vectorsNumber; row++)
                  if (degree[row] > tempMaxCliqueDegree) {
                        tempMaxCliqueDegree = degree[row];
                  }
            System.out.println("Potential Max Clique Degree: "
                        + (tempMaxCliqueDegree + 1) + ".");

            initialTempMaxCliqueDegree = tempMaxCliqueDegree;

            // 4. Let v_k the vertices with degree greater than or equal
to (PMC#-1),for some k, 1<=k<=n. Find the # of vertices v_k with degree
greater than or equal to (PMC#-1), say “inducedSum”.
            do {
                  int inducedSum = 0;// the # of vertices with degree
greater than or equal to (PMC#-1)
                  for (row = 0; row < vectorsNumber; row++) {
                        if (degree[row] >= tempMaxCliqueDegree) {
                              booleanDegree[row] = 1;
                              inducedSum       =       inducedSum       +
booleanDegree[row];
                        } else
                              booleanDegree[row] = 0;
                  }

                  // 4a. If “inducedSum” is greater than or equal to
PMC#, then maybe there is a Max Clique with Clique# = PMC#.
                  if (inducedSum >= tempMaxCliqueDegree + 1) {
                        System.out.println("Maybe there is a K"
                        + (tempMaxCliqueDegree + 1) + " Max Clique.");
                        System.out.println();
                        initialMaxClique = true;

                        // 4b. Else if “inducedSum” is NOT greater than
                  or equal to PMC#, there is not a Maximum Clique with
                  Clique# = PMQ#. Set PMQ#=PMQ#-1 and continue by going
                  back to step 4.
                  } else {
                        System.out.println("There is not a K"
                        + (tempMaxCliqueDegree + 1) + " Max Clique.");
                        System.out.println("Check for K"
                        + tempMaxCliqueDegree)+ " Max Clique.");
                        System.out.println();
                        tempMaxCliqueDegree--;
                  }
            }  while  (tempMaxCliqueDegree  >  0  &&  initialMaxClique  ==
false);
            //
************************************************************************
*********************************************

            // II. Search for the Max Clique, based on the Upper Bound

            newTempMaxCliqueDegree  =  tempMaxCliqueDegree;//  This  is
equal to (Potential Maximum Clique # -1), (PMC#-1).
            // Its value decreases at the end of the algorithm if all
the intermediate conditions are NOT met, until Maximum Clique is found.
            // Then Clique#=(updated by the algorithm)PMC#

            int[] adjacentVector = new int[initialTempMaxCliqueDegree +
1];// The vector where the vertices of the closed neighborhood of each
v_k are stored.
            int[]        booleanDegreeOfInducedTable        =        new
int[initialTempMaxCliqueDegree + 1];
            int[]          degreeOfInducedTable          =          new
int[initialTempMaxCliqueDegree + 1];

            //  1.  For  every  v_k  with  degv_k=(PMC#-1),  investigate  if
<N[Vk]> is a Max Clique with Clique# = PMQ#:
            do {
                  System.out.println("Check       for       K"       +
(newTempMaxCliqueDegree + 1)
                              + " Max Clique.");
                  System.out.println();
                  int countForMaxClique = 0;
                  int vectorOfMaxClique = 0;

                  // a. Build <N[V_k]>:
                  do {
                        int  count  =  0;//  This  is  the  cardinality  of
v_k's
                        // neighborhood set.
                        vectorOfMaxClique = countForMaxClique;
                        adjacentVector[0] = countForMaxClique;
                        System.out.println();
                        System.out.println("Check if the vertex V"
                        + (countForMaxClique + 1) + " belongs to a K"
                       + (newTempMaxCliqueDegree + 1) + " Max Clique.");

                        adjacentVector[count] = countForMaxClique;

                        // i. First, find the adjacent vertices to v_k.
                        // ii. Then, store v_k and its adjacent vertices
in a one dimensional array, say [V_k] (adjacentVector[]).
                        for  (column  =  0;  column  <  vectorsNumber;
column++) {
                              if (table[vectorOfMaxClique][column] == 1)
{
                                    count++;
                                    adjacentVector[count] = column;
                              }
                        }

                        // b. Check if the degrees of the vertices of
<N[V_k]> are greater than or equal to the (PMQ#-1):
                        // i. First, find the degree in <N[V_k]> of each
vertex of <N[V_k]>
                        for (row = 0; row < count + 1; row++) {
                              int  sum1  =  0;//  This  is  the  degree  in
<N[V_k]> of each vertex of <N[V_k]>
                              for  (column  =  0;  column  <  count  +  1;
column++)
                                    sum1 = sum1 +
table[adjacentVector[row]][adjacentVector[column]];
                              degreeOfInducedTable[row] = sum1;
                        }

                        // ii. Then, find the # of vertices with degree
greater than or equal to the (PMQ#-1), say "inducedSum1".
                        int inducedSum1 = 0;//This is the # of vertices,
of <N[V_k]> , with degree greater than or equal to the (PMQ#-1).
                        for (row = 0; row < count + 1; row++) {
                              if      (degreeOfInducedTable[row]      >=
newTempMaxCliqueDegree) {
                                    booleanDegreeOfInducedTable[row]   =
1;
                                    inducedSum1++;
                              } else
                                    booleanDegreeOfInducedTable[row]   =
0;
                              // inducedSum1 = inducedSum1 +
                              // booleanDegreeOfInducedTable[row];
                        }
                        // c. If the "inducedSum1" is greater than or
equal to the PMQ# then the Clique# = PMQ#.
                        if (inducedSum1 >= newTempMaxCliqueDegree + 1) {
```

```
                                    System.out.println("YES!!! V"      +
(countForMaxClique   +   1)   +   "  belongs  to  at  least  one  K"
+(newTempMaxCliqueDegree + 1) + " Max Clique.");
                                    System.out.println();

                                    int rowUpTriangular = 1;
                                    int columnUpTriangular = 1;

                                    // Find the Maximum Cliques that v_k
belongs   to.   Consider   the   one   dimensional   array   [V_k],
(“adjacentVector[]”),  which  contains  the  elements  of  the  closed
neighborhood of v_k, N[V_k].
                                    // *******
                                    //  Also,  consider  each  element  of
v_k’s neighborhood, say v_m, for some m, 1<=m<=(cardinality of v_k's
                                    // neighborhood set).
                                    for  (row  =  rowUpTriangular;  row  <
count + 1; row++) {

                                    int top  = -1;// counts the top for
the vector with the vertices adjacent to the "second vertex"
                                            int topMaxClique = 2;//
counts the top for the vector with the Max Clique vertices
                                            int[]   stack   =   new
int[initialTempMaxCliqueDegree  +  1];//  This  is  the  vector  where  the
vertices of the neighborhood of each element of N[V_k] are stored.

                                            int[]  stackMaxClique  =
new int[initialTempMaxCliqueDegree + 1];// This is the vector where the
vertices consisting a Maximum Clique are stored.
                                            System.out.println();
                                            System.out.println("the
element v_m now is: " + (adjacentVector[row] + 1));
                                            System.out.println();

                                            //  1)  For  each  element
of v_k's neighborhood,say v_m, for some m, 1<=m<=(cardinality of v_k's
neighborhood set), find its neighborhood N(V_m) in <N[V_k]> and store it
in a one dimensional array, say “stack”.
                                            for      (column      =
columnUpTriangular; column < count + 1; column++) {
                                                  if
(table[adjacentVector[row]][adjacentVector[column]] == 1) {
                                                        top++;
                                                        stack[top] =
adjacentVector[column];
      System.out.println("stack[" + top + "]= " + stack[top]);
                                                  }
                                            }
      System.out.println("top= " + top);
                                            System.out.println();

                                            // 2) Create another one
dimensional  array,  say  “stackMaxClique”.  This  is  the  vector  where  the
vertices,  consisting  a  Maximum  Clique,  are  stored.  Store  v_k  as  first
```

```
element, v_m as second element (starting with m=1) and the first element
of the “stack”, as third element (these 3 vertices consist a K3).
                                                stackMaxClique[0]      =
countForMaxClique;
                                                stackMaxClique[1]      =
adjacentVector[row];
                                                stackMaxClique[2]      =
stack[0];

//   3)   Set   “elementA”=the   first   element   of   the   “stack”,   and
“elementB”=the  second  one.  Check  if  “elementA”  and  “elementB”  are
adjacent:
                                                int countTop = 0;
                                                int stackElementA = 0;
                                                int stackElementB = 1;
                                                boolean stop = false;

                                                if (top < 1) {

      System.out.println("For v_k= " + (countForMaxClique + 1) + " and
v_m= " + (adjacentVector[row] + 1) + " there is NOT a Max Clique.");
                                                }

                                                if (top >= 1) {

                                                      do {
                                                            if
(table[stack[stackElementA]][stack[stackElementB]] == 1) {

                                                                  int
countInTheStackMaxClique = 0;

                                                                  //  a)
// If yes, check if the “elementB” is adjacent with each element of
“stackMaxClique”   (start   checking   by   the   third   element   of
“stackMaxClique”).
                                                                  for
(int i = 0; i <= topMaxClique; i++) {

      if (table[stackMaxClique[i]][stack[stackElementB]] == 1) {

      countInTheStackMaxClique++;

      }
                                                                  }

      System.out.println("topMaxClique= " + topMaxClique);

      System.out.println("countInTheStackMaxClique= "
      + countInTheStackMaxClique);

// i. If yes, then add “elementB” to “stackMaxClique”, and go back to
step 4), where “elementA”=“elementB” and “elementB”= the next element of
“stack”.
```

```
                                                                  if
(countInTheStackMaxClique == topMaxClique + 1) {

      topMaxClique++;
      stackMaxClique[topMaxClique] = stack[stackElementB];
      stackElementA = stackElementB;
      stackElementB++;
      System.out.println("stackMaxClique[" + topMaxClique + "]= "
      + stackMaxClique[topMaxClique]);

      countTop++;
                                                                  }

// ii. If no then go back to step 4), where “elementA” remains the same
and “elementB”=the next element of “stack”.
                                                                  else {
      stackElementB++;
                                                                  }
// This is the condition to terminate the do-while loop.
                                                                  if
((stackElementA == top && stackElementB == (top + 1))
|| (stackElementA == (top - 1) && stackElementB == (top + 1))) {

      stop = true;
                                                                  }
                                                            }

// b) If no go back to step 4), where “elementA” remains the same and
“elementB”=the next element of “stack”.
                                                            else {

      stackElementB++;
                                                            }

                                                            //   c)   If
check all elements of “stack”, as “elementB” then “elementA”=the next
element of “stack” and “elementB”=the element of “stack” which is after
the new “elementA”.
                                                            if
(stackElementB == top + 1) {
      stackElementA++;
      stackElementB = stackElementA + 1;
                                                            }
      System.out.println("stop= " + stop);

// This is the condition to terminate the do-while loop.

                                                            if
((stackElementA == top && stackElementB == (top + 1))
|| (stackElementA == (top - 1) && stackElementB == (top + 1))) {
                                                                  stop =
true;
                                                            }

      System.out.println("stackElementA= " + stackElementA);
      System.out.println("stackElementB= " + stackElementB);
                                                         }  while  (stop  ==
false);
                                    if(topMaxClique>maxCliqueDegree){
                                    maxCliqueDegree=topMaxClique+1;
                                    }

                              for (int j = 0; j <= topMaxClique; j++) {
                              maximalClique[countMaximalCliques][j]=
                              stackMaxClique[j] + 1;
                              }

                              countMaximalCliques++;


       System.out.print("The Max Clique is: ");
                                                      for  (int  i  =  0;  i
<= topMaxClique; i++) {

            System.out.print("V" + (stackMaxClique[i] + 1) + " ");
                                                      }
                                                }
                                                System.out.println();
                                          }
                                    }

                                    // d) Continue with the next element
of N[V_k], v_(m+1), going back to step 1).
                              }

//**********************************************************************
*******************************
                              // v. Else if the “inducedSum1” is equal
to the PMQ# then v_k belongs to one Max Clique with Clique# = PMQ#.
                              else       if       (inducedSum1       ==
newTempMaxCliqueDegree + 1) {
                                    System.out.println("YES!!!  There  is
a K" + (newTempMaxCliqueDegree + 1) + " Max Clique.");
                                    maxClique = true;
                                    maxCliqueDegree                    =
newTempMaxCliqueDegree + 1;
                                    System.out.print("The Max Clique is:
V" + (countForMaxClique + 1) + " ");

                                    for (int i = 1; i <= count; i++)
                                          if (degreeOfInducedTable[i] >=
newTempMaxCliqueDegree) {
                                                System.out.print("V"    +
(adjacentVector[i] + 1) + " ");
                                          }
                                    System.out.println();
                              }
                        }
                        // d. If the # of vertices of <N[V_k]> are not
greater than or equal to the PMQ# then continue searching for a Clique#
= PMQ# by going back to step II.1. with the next vertex v_(k+1)
                        else
                              System.out.println("The    vertex    V"    +
(countForMaxClique    +    1)    "    does    not    belong    to    a    K"
(newTempMaxCliqueDegree + 1) + " Max Clique. ");

                        countForMaxClique++;

                  } while (countForMaxClique < vectorsNumber);
                  System.out.println();

                  // II.2. If there is not a Clique# = PMQ# then set
PMQ# = PMQ#-1 and continue searching for a Maximum Clique with Clique# =
(PMC#-1) by going back to step I.4.
                  newTempMaxCliqueDegree--;

            } while (newTempMaxCliqueDegree > 0 && maxClique == false);

            // print the outputs
            System.out.println("The Maximal Cliques are:");
            for (int i = 0; i < countMaximalCliques; i++) {
                  System.out.println();
            for (int j = 0; j < maxCliqueDegree; j++) {
                  System.out.print(maximalClique[i][j]+" ");
            }

            }
            System.out.println();

            System.out.println("The Maximum Clique is K" +
maxCliqueDegree);

      }
}
```

THE OUTPUT

```
Maximum Clique Problem of G(37,85)

Find the degree of each vertex
degree(v1 )= 2
degree(v2 )= 4
degree(v3 )= 4
degree(v4 )= 3
degree(v5 )= 4
degree(v6 )= 2
degree(v7 )= 2
degree(v8 )= 6
degree(v9 )= 3
degree(v10 )= 4
degree(v11 )= 3
degree(v12 )= 6
degree(v13 )= 3
degree(v14 )= 3
degree(v15 )= 3
degree(v16 )= 1
degree(v17 )= 14
degree(v18 )= 5
degree(v19 )= 3
degree(v20 )= 10
degree(v21 )= 7
degree(v22 )= 10
degree(v23 )= 15
degree(v24 )= 3
degree(v25 )= 4
degree(v26 )= 4
degree(v27 )= 2
degree(v28 )= 8
degree(v29 )= 5
degree(v30 )= 10
degree(v31 )= 3
degree(v32 )= 4
degree(v33 )= 1
degree(v34 )= 1
degree(v35 )= 4
degree(v36 )= 2
degree(v37 )= 2

Potential Max Clique Degree: 16.
There is not a K16 Max Clique.
Check for K15 Max Clique.

There is not a K15 Max Clique.
Check for K14 Max Clique.

There is not a K14 Max Clique.
Check for K13 Max Clique.

There is not a K13 Max Clique.
Check for K12 Max Clique.
```

```
There is not a K12 Max Clique.
Check for K11 Max Clique.

There is not a K11 Max Clique.
Check for K10 Max Clique.

There is not a K10 Max Clique.
Check for K9 Max Clique.

There is not a K9 Max Clique.
Check for K8 Max Clique.

There is not a K8 Max Clique.
Check for K7 Max Clique.

Maybe there is a K7 Max Clique.

Check for K7 Max Clique.

Check if the vertex V1 belongs to a K7 Max Clique.
The vertex V1 does not belong to a K7 Max Clique.

Check if the vertex V2 belongs to a K7 Max Clique.
……………………………………………………………………………………………………………………………………………
……………………………………………………………………………………………………………………………………………
The vertex V37 does not belong to a K7 Max Clique.

Check for K6 Max Clique.

Check if the vertex V1 belongs to a K6 Max Clique.
The vertex V1 does not belong to a K6 Max Clique.

Check if the vertex V2 belongs to a K6 Max Clique.
…………………………………………………………………………………………………………………………………………
…………………………………………………………………………………………………………………………………………

Check if the vertex V17 belongs to a K6 Max Clique.
YES!!! There is a K6 Max Clique.
The Max Clique is: V17 V18 V21 V22 V23 V28

……………………………………………………………………………………………………
……………………………………………………………………………………………………
The Maximum Clique is K6
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability. A guide to the Theory of NP-Completeness,* Paperback, New York, 1979.

[2] P. R. J. Ostergard, *A fast algorithm for the maximum clique problem,* Discrete Applied Mathematics Volume 120, Issues 1-3, pp. 197-207, 15 August 2002.

[3] L. Babel, *Finding maximum cliques in arbitrary and in special graphs,* computing 46, pp. 321-341, 1991.

[4] D. S. Johnson and M. A. Trick, *Cliques, Coloring and Satisfiability,* Second DIMACS Challenge, October 11-13, 1993, Volume 26, American Mathematical Society, 1996.

[5] B. Balasundaram et al., *Clique Relaxations in Social Network Analysis: The Maximum k-plex Problem,* 12 Dec 2006.

[6] K. J. Bell, *Implementation of an efficient algorithm to detect maximal cliques in a conflict graph,* Master's Thesis, Naval Postgraduate School, Monterey, California, June 1990.

[7] R. Tarjan , *Finding a maximum clique,* March 1972.

[8] A. Dharwadker, *The Clique Algorithm,* 2006.

[9] S. Homer and M. Peinado, *On the Performance of Polynomial-time CLIQUE Approximation Algorithms on Very Large Graphs.* Second DIMACS Implementation Challenge, October 11-13, 1993, Volume 26, pp. 103-124, American Mathematical Society, 1996.

[10] S. Wasserman and K. Faust, *Social Network Analysis,* Cambridge University Press, 1994.

[11] G. Chartrand and P. Zhang, *Introduction to Graph Theory,* McGraw Hill, 2005.

[12] D. R. Wood, *An algorithm for finding a maximum clique in a graph,* Operations Research Letters, 21, pp. 211-217, 1997.

[13] R. E. Tarjan, A. E. Trojanowski, *Finding a maximum independent set,* SIAM J. Comput. 6(3), pp. 537-546, 1977.

[14] E. Balas, J. Xue, *Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring,* Algorithmica 15, pp. 397-412, 1996.

[15] P. M. Pardalos, J. Xue, *The maximum clique problem*, J. Global Optim. 4, pp. 301-328, 1994.

[16] V. E. Krebs, *Mapping networks of terrorist cells.* Connections 24(3): pp. 43-52, 2002.

[17] P. H. Fellman and R. Wright, *Modeling Terrorist Networks - Complex Systems at the Mid-Range,* paper prepared for the Joint Complexity Conference, London School of Economics, September 16-18, 2003.

[18] P. V. Fellman, J. P. Clemens, R. Wright, J. V. Post and M. Dadmun, *Disrupting Terrorist Networks- A Dynamic Fitness Landscape Approach,* Interjournal Complex System, 2060, 2007.

[19] N. Memon and H. L. Larsen, *Structural Analysis and Destabilizing Terrorist Networks*, Proceedings of the 2006 International Conference On Data Mining (DMIN 2006), pp. 296-302, 2006.

[20] A. Degenne and M. Forse, *Introducing Social Networks.* London: Sage Publications, 1999.

[21] D. M. Akbar Hussain, *Terrorist Networks Analysis through Argument Driven Hypothesis Model,* Second International Conference on Availability, Reliability and Security (ARES '07), 2007 IEEE.

[22] R. Alba, *A Graph-TheoreticDefinition of a Sociometric Clique,* Journal of Mathematical Sociology 3, pp. 113-126, 1973.

[23] L. C. Freeman, *The Sociological Concept of "group": An empirical test of two models,* American Journal of Sociology 98, pp. 152-166, 1992.

[24] S. B. Seidman and B. L. Foster, *A Graph Theoretic Generalization of the Clique Concept*, Journal of Mathematical Sociology 6, pp. 139-154, 1978.

[25] *Stack*, NIST,*http://www.nist.gov/dads/HTML/stack.html*, 21 May, 2008.

[26] B. Champagne, *Anatomy of a Terrorist Attack: An in-Depth Invetigation into the 1998 Bombings of the US Embassies in Kenya and Tanzania.* Working Paper, Ridgway Center, International Security Studies, University of Pittsburg, Pittsburg PA, Spring 2005.

[27] J. L. Geffre, *A Layered Social and Operational Network Analysis,* Master's Thesis, Air Force Institute of Technology, Ohio, March 2007.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Matthew. Carlyle
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

4. Dr. Ralucca Gera
   Department of Applied Mathematics
   Naval Postgraduate School
   Monterey, California

5. Dr. Pantelimon Stanica
   Department of Applied Mathematics
   Naval Postgraduate School
   Monterey, California

6. Dr. Clyde Scandrett
   Chairman, Department of Applied Mathematics
   Naval Postgraduate School
   Monterey, California

7. Hellenic Navy General Staff
   Athens, Greece

8. LCDR Spyridon Pollatos
   Naval Postgraduate School
   Monterey, California